

Parameterized Algorithms for Graph Partitioning Problems

Anjeneya Swami Kare

A Thesis Submitted to
Indian Institute of Technology Hyderabad
In Partial Fulfillment of the Requirements for
The Degree of Doctor of Philosophy




Department of Computer Science and Engineering

May 2018

Declaration

I declare that this written submission represents my ideas in my own words, and where ideas or words of others have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.

 26/07/2018

(Signature)

(Anjeneya Swami Kare)

CS14RESCH01002

(Roll No.)

Approval Sheet

This Thesis entitled Parameterized Algorithms for Graph Partitioning Problems by Anjeneya Swami Kare is approved for the degree of Doctor of Philosophy from IIT Hyderabad

S.P.Pal: 26/07/18

(Prof. Sudebkumar Prasant Pal) External Examiner
Dept. of Computer Science and Engineering
IIT Kharagpur

N. S. Narayanaswamy 26/07/18

(Prof. N S Narayanaswamy) External Examiner
Dept. of Computer Science and Engineering
IIT Madras

Karteek Sreenivasaiah

(Dr. Karteek Sreenivasaiah) Internal Examiner
Dept. of Computer Science and Engineering
IIT Hyderabad

N. R. Aravind

(Dr. N. R. Aravind) Advisor
Dept. of Computer Science and Engineering
IIT Hyderabad

Subrahmanyam Kalyanasundaram 26/07/18

(Dr. Subrahmanyam Kalyanasundaram) Advisor
Dept. of Computer Science and Engineering
IIT Hyderabad

J.P.L

(Dr. Phanindra Jampana) Chairman
Dept. of Chemical Engineering
IIT Hyderabad

Acknowledgements

Firstly, I would like to express my heartfelt thanks and gratitude to Dr. N. R. Aravind and Dr. Subrahmanyam Kalyanasundaram for being my supervisors. I must confess that I am lucky enough to have such supervisors as they gave me freedom to choose research problems that I ultimately was able to follow and constitute my own interests. I am indebted to both of them for sparing kind enough of time to have invigorating discussions that, at times, gave crucial directions to this thesis. I would like to express my greatest appreciation for their care and attention in knowing the status of my work particularly when there was an occurrence of delay from my end. They were generous enough to grant funds from their personal research fund to pay registration fee of a conference for which I owe my deepest gratitude.

I would like to offer my sincere thanks to Dr. N. R. Aravind for an umpteen number of technical discussions I had with him which were crucial for the results in this thesis. I thank him for motivating me in the form of giving time to discuss articles related to exact and parameterized algorithms, thus he is responsible for pushing me to explore more and to acquire knowledge and develop an interest in this area. His crucial insights always used to put me on heels and thus he helped me to imbibe this nature of constant improvement. At a point of time, he noticed that I was just reading and was not attempting to pose research questions. This cognizance of him, indeed, gave an opportunity to locate my weakness which changed my perspective on posing research questions, else this thesis wouldn't have been possible.

I would like to express my honest gratitude to Dr. Subrahmanyam Kalyanasundaram with whom I have developed a scholarly rapport. On many occasions I had technical discussions with him which eventually crucial to the ideas present in this thesis. I am thankful to him for giving lot of his time in preparing the technical content to submit the papers to different conferences and journals. Many a times, he took time and pains to meet in my office at HCU for discussions whenever we could not meet up in IITH. I pay my heartfelt thanks to him for giving me an opportunity to review couple of research papers, thus, I was introduced to the realm of the process of research paper reviewing. I also indebted to him for making me part of his other research scholars work so that I was able to learn the nuances of research and he also kept a continuous positive light in me by the way of talking positive things about my work which lead me to see the light.

I would like to thank one of my co-authors Dr. Juho Lauri with whom I had plenty of discussions over email on happy coloring problems. Some of the results on happy coloring problems were originated during these discussions. During this process I also got to know some of the concepts of parameterized complexity which was very crucial for this thesis. He was also very supportive in preparing the technical content for a research

article on happy coloring problems. I thank Dr. Sandeep R. B. for giving his time on many occasions for discussing research articles. I thank my fellow PhD student Sriram Bhyravarapu for keeping me part of his research work and assisting each other whenever possible.

I thank the department of CSE, IITH for giving me an opportunity to pursue PhD in IITH. I thank the former heads of the department Dr. C. Krishna Mohan, Dr. Bheemarjuna Reddy Tamma and the current head of the department Dr. M. V. Panduranga Rao for their support. I thank my DRC committee members for giving valuable suggestions at different stages of my PhD. I thank Dr. Ramakrishna Upadrasta for his encouragement in many occasions which gave lot confidence and motivation to perform well. I thank all the faculty and staff of CSE department, whose names I could not mention here, for their unconditional support throughout my doctoral study. I thank the staff at academic and accounts offices for helping me in many academic and financial matters. I thank IITH for providing congenial environment and providing me financial support for attending an international conference.

I thank University of Hyderabad for granting me study leave to pursue PhD. I thank Prof. Arun K. Pujari former dean of SCIS who encouraged me to take up leave and pursue full time PhD. He was very supportive in approving my study leave. I thank Dr. Anupama Potluri who was constantly encouraging me to do well in my PhD. I thank all the faculty and staff of SCIS who has supported me directly or indirectly during my PhD.

I thank my friend Dr. Nookaraju Bendukurthi who is my pillar of strength. He is the one who is been encouraging me to come to academics and to pursue PhD. I thank my friends Naveen Davuluri, Lakshman Rao Vittanala, Nagesh Devireddy, Sateesh Pabboju who were encouraging me to do well in research.

Last but not the least, I thank my family, my parents for their unconditional love and encouragement, my brother who always there to cherish my achievements as his, my wife for her financial and moral support all through my PhD journey and my daughter who is a stress buster for me. Finally I thank the almighty for continuously showering positive things and keeping positive people around me.

Abstract

In parameterized complexity, a problem instance (I, k) consists of an input I and an extra parameter k . The parameter k usually a positive integer indicating the size of the solution or the structure of the input. A computational problem is called *fixed-parameter tractable* (FPT) if there is an algorithm for the problem with time complexity $O(f(k) \cdot n^c)$, where $f(k)$ is a function dependent only on the input parameter k , n is the size of the input and c is a constant. The existence of such an algorithm means that the problem is tractable for fixed values of the parameter. In this thesis, we provide parameterized algorithms for the following NP-hard graph partitioning problems:

(i) **MATCHING CUT Problem:** In an undirected graph, a matching cut is a partition of vertices into two non-empty sets such that the edges across the sets induce a matching. The matching cut problem is the problem of deciding whether a given graph has a matching cut. The **MATCHING CUT** problem is expressible in monadic second-order logic (MSOL). The MSOL formulation, together with Courcelle's theorem implies linear time solvability on graphs with bounded tree-width. However, this approach leads to a running time of $f(\|\varphi\|, t) \cdot n$, where $\|\varphi\|$ is the length of the MSOL formula, t is the tree-width of the graph and n is the number of vertices of the graph. The dependency of $f(\|\varphi\|, t)$ on $\|\varphi\|$ can be as bad as a tower of exponentials.

In this thesis we give a single exponential algorithm for the **MATCHING CUT** problem with tree-width alone as the parameter. The running time of the algorithm is $2^{O(t)} \cdot n$. This answers an open question posed by Kratsch and Le [Theoretical Computer Science, 2016]. We also show the fixed parameter tractability of the **MATCHING CUT** problem when parameterized by neighborhood diversity or other structural parameters.

(ii) **H -FREE COLORING Problems:** In an undirected graph G for a fixed graph H , the **H -FREE q -COLORING** problem asks to color the vertices of the graph G using at most q colors such that none of the color classes contain H as an induced subgraph. That is every color class is H -free. This is a generalization of the classical **q -COLORING** problem, which is to color the vertices of the graph using at most q colors such that no pair of adjacent vertices are of the same color. The **H -FREE CHROMATIC NUMBER** is the minimum number of colors required to H -free color the graph.

For a fixed q , the **H -FREE q -COLORING** problem is expressible in monadic second-order logic (MSOL). The MSOL formulation leads to an algorithm with time complexity $f(\|\varphi\|, t) \cdot n$, where $\|\varphi\|$ is the length of the MSOL formula, t is the tree-width of the graph and n is the number of vertices of the graph.

In this thesis we present the following explicit combinatorial algorithms for **H -FREE COLORING** problems:

- An $O(q^{O(tr)} \cdot n)$ time algorithm for the general H -FREE q -COLORING problem, where $r = |V(H)|$.
- An $O(2^{t+r \log t} \cdot n)$ time algorithm for K_r -FREE 2-COLORING problem, where K_r is a complete graph on r vertices.

The above implies an $O(t^{O(tr)} \cdot n \log t)$ time algorithm to compute the H -FREE CHROMATIC NUMBER for graphs with tree-width at most t . Therefore H -FREE CHROMATIC NUMBER is FPT with respect to tree-width.

We also address a variant of H -FREE q -COLORING problem which we call H -(SUBGRAPH)FREE q -COLORING problem, which is to color the vertices of the graph such that none of the color classes contain H as a subgraph (need not be induced).

We present the following algorithms for H -(SUBGRAPH)FREE q -COLORING problems.

- An $O(q^{O(tr)} \cdot n)$ time algorithm for the general H -(SUBGRAPH)FREE q -COLORING problem, which leads to an $O(t^{O(tr)} \cdot n \log t)$ time algorithm to compute the H -(SUBGRAPH)FREE CHROMATIC NUMBER for graphs with tree-width at most t .
- An $O(2^{O(t^2)} \cdot n)$ time algorithm for C_4 -(SUBGRAPH)FREE 2-COLORING, where C_4 is a cycle on 4 vertices.
- An $O(2^{O(tr-2)} \cdot n)$ time algorithm for $\{K_r \setminus e\}$ -(SUBGRAPH)FREE 2-COLORING, where $K_r \setminus e$ is a graph obtained by removing an edge from K_r .
- An $O(2^{O((tr^2)^{r-2})} \cdot n)$ time algorithm for C_r -(SUBGRAPH)FREE 2-COLORING problem, where C_r is a cycle of length r .

(iii) Happy Coloring Problems: In a vertex-colored graph, an edge is *happy* if its endpoints have the same color. Similarly, a vertex is *happy* if all its incident edges are happy. we consider the algorithmic aspects of the following MAXIMUM HAPPY EDGES (k -MHE) problem: given a partially k -colored graph G , find an extended full k -coloring of G such that the number of happy edges are maximized. When we want to maximize the number of happy vertices, the problem is known as MAXIMUM HAPPY VERTICES (k -MHV).

We show that both k -MHE and k -MHV admit polynomial-time algorithms for trees. We show that k -MHE admits a kernel of size $k + \ell$, where ℓ is the natural parameter, the number of happy edges. We show the hardness of k -MHE and k -MHV for some special graphs such as split graphs and bipartite graphs. We show that both k -MHE and k -MHV are tractable for graphs with bounded tree-width and graphs with bounded neighborhood diversity.

In the last part of the thesis we present an algorithm for the Replacement Paths Problem which is defined as follows: Let G ($|V(G)| = n$ and $|E(G)| = m$) be an undirected graph with positive edge weights. Let $P_G(s, t)$ be a shortest $s - t$ path in G . Let l be the number of edges in $P_G(s, t)$. The EDGE REPLACEMENT PATH problem is to compute a shortest $s - t$ path in $G \setminus \{e\}$, for every edge e in $P_G(s, t)$. The NODE REPLACEMENT PATH problem is to compute a shortest $s - t$ path in $G \setminus \{v\}$, for every vertex v in $P_G(s, t)$.

We present an $O(T_{SPT}(G) + m + l^2)$ time and $O(m + l^2)$ space algorithm for both the problems, where $T_{SPT}(G)$ is the asymptotic time to compute a single source shortest path tree in G . The proposed algorithm is simple and easy to implement.

Contents

Declaration	ii
Approval Sheet	iii
Acknowledgements	iv
Abstract	vi
1 Introduction	4
1.1 MATCHING CUT Problem	6
1.1.1 Previous Work	7
1.1.2 Our Results	8
1.2 H -Free Coloring Problems	8
1.2.1 Related Work	9
1.2.2 Our Results	9
1.3 Happy Coloring Problems	10
1.3.1 Previous Work	11
1.3.2 Our Results	12
1.4 Replacement Paths Problems	13
1.4.1 Previous Work	14
1.4.2 Our Results	15
2 Preliminaries	16
2.1 Graph Terminology	16
2.2 Parameterized Complexity	19
2.3 Tree-Width	20
2.4 Branch-width	21
2.5 MSOL and Courcelle’s Theorem	22
2.6 Neighborhood Diversity	23
2.7 Problem Definitions	24
2.8 Other Notations	27
2.9 Organization	27

3	Algorithms for Matching Cut Problem	28
3.1	Graphs with Bounded Tree-width	28
3.2	Graphs with Bounded Neighborhood Diversity	31
3.3	Other Structural Parameters	32
4	Algorithms for H-Free Coloring Problems	34
4.1	Overview of the Techniques Used	35
4.2	H -Free Coloring	36
4.2.1	K_r -FREE 2-COLORING	36
4.2.2	H -FREE 2-COLORING	37
4.2.3	H -FREE q -COLORING	42
4.3	H -(Subgraph)Free Coloring	46
4.3.1	C_4 -(SUBGRAPH)FREE 2-COLORING	46
4.3.2	$\{K_r \setminus e\}$ -(SUBGRAPH)FREE 2-COLORING	48
4.3.3	C_r -(SUBGRAPH)FREE 2-COLORING	51
4.3.4	H -(SUBGRAPH)FREE 2-COLORING	56
4.3.5	H -(SUBGRAPH)FREE q -COLORING	57
5	Happy Coloring Problems	58
5.1	Algorithm for k -MHV Problem for Trees	58
5.1.1	Computing $T_v[i, H]$	60
5.1.2	Computing $T_v[i, U]$	61
5.1.3	Generating all optimal happy vertex colorings	63
5.2	Algorithm for k -MHE problem for Trees	64
5.2.1	Generating all optimal happy edge colorings	66
5.3	MHV and MHE on Complete Graphs	66
5.4	Hardness Results for Happy Coloring Problems	67
5.4.1	k -MHE for planar graphs and graphs with bounded branch width	69
5.5	Exact Exponential-Time Algorithms for Happy Coloring	70
5.6	A Linear Kernel for WEIGHTED MHE	72
5.6.1	Polynomial Time Algorithm for Subproblems of WEIGHTED MHE	72
5.7	Structural parameterization	76
5.7.1	Tree-Width	76
5.7.2	Neighborhood Diversity	77
6	Algorithm for Replacement Paths Problem	81
6.1	Edge Replacement Paths	81
6.1.1	Labeling the nodes of G	83

6.1.2	Computing Swap Edges	83
6.2	Node Replacement Paths	85
7	Conclusions and Future Work	88
	References	90

List of Figures

2.1	An example graph.	21
2.2	Tree decomposition of the graph G	21
2.3	A nice tree decomposition of G	21
4.1	An example graph H	37
4.2	Forming H at an introduce node. Sequence $s = (v, v_2, v_1, \text{FG}, \text{FG}, \text{FG})$	37
4.3	Forming H at join node. Sequences at node j_1 $s' = (\text{DC}, \text{DC}, v_1, v_2, \text{FG}, \text{FG})$, at node j_2 $s'' = (\text{FG}, \text{FG}, v_1, v_2, \text{DC}, \text{DC})$ gives a sequence $s = (\text{FG}, \text{FG}, v_1, v_2, \text{FG}, \text{FG})$ at node i . Vertices outside the dashed lines are forgotten vertices.	38
4.4	A cycle of length 15 formed with vertex v at an insert node. The vertices outside the dotted outline are forgotten vertices. The ℓ values are marked between the u_i 's in the figure.	51
4.5	A cycle of length 20 formed using the paths from left and right subtrees at join node. The vertices outside the dotted outline are forgotten vertices.	51
5.1	(a) A graph G of an instance of DMHE, where white vertices correspond to uncolored vertices. (b) The graph G transformed into a split graph G' by the construction of Theorem 25. The edges between the vertices in C are not drawn.	68
5.2	A set of a type partition, where each vertex in $Q_1 \cup Q_2$ has the same type. The dashed edges appear exactly when $Q_1 \cup Q_2$ induces a clique. The set Q_1 forms a complete bipartite graph with both X_1 and X_2 ; likewise for Q_2 (edges omitted for brevity).	78
6.1	Potential replacement paths for the edge e_i . The zig-zag lines represent a path	82

6.2 (a) An SPT rooted at s . Solid lines are part of the SPT. Dashed lines represent the non-tree edges (we omit the edge weights). Number inside the vertex circle denotes the vertex number, where as the number above the vertex circle represents vertex label. (b) Corresponding RSP-DAG with set of non-tree edges associated with nodes 84

List of Publications

Papers [4, 5, 47] appeared as conference proceedings. Papers [6, 7] are under review. Paper [46] is not part of the thesis.

- [4] N. R. Aravind, Subrahmanyam Kalyanasundaram and **Anjeneya Swami Kare**. Linear Time Algorithms for Happy Vertex Coloring Problems for Trees. In 27th International Workshop on Combinatorial Algorithms IWOCA 2016 pages 281–292, 2016.
- [5] N. R. Aravind, Subrahmanyam Kalyanasundaram and **Anjeneya Swami Kare**. On Structural Parameterizations of the Matching Cut Problem. In 11th International Conference on Combinatorial Optimization and Applications COCOA 2017 pages 475–482, 2017.
- [47] **Anjeneya Swami Kare**. A Simple Algorithm For Replacement Paths Problem. In International Conference on Graph Theory and its Applications ICGTA 2015. Appeared in Electronic Notes in Discrete Mathematics (53), Elsevier, pages 307-318 (2016).
- [6] Akanksha Agrawal, N. R. Aravind, Subrahmanyam Kalyanasundaram, **Anjeneya Swami Kare**, Juho Lauri, Neeldhara Misra and I. Vinod Reddy. Parameterized Complexity of Happy Coloring Problems (Under Review).
- [7] N. R. Aravind, Subrahmanyam Kalyanasundaram and **Anjeneya Swami Kare**. H -Free Coloring on Graphs with Bounded Tree-Width (Under Review).
- [46] Saurabh Joshi, Subrahmanyam Kalyanasundaram, **Anjeneya Swami Kare** and Sriram Bhyravarapu. On the Tractability of (k, i) -Coloring. Appeared in 4th International Conference on Algorithms and Discrete Applied Mathematics CALDAM 2018.

Chapter 1

Introduction

Graph theory has played an important role in solving many real world computational problems. Mathematicians and computer scientists formulate many of these computational problems as algorithmic problems on graphs so that they can design efficient algorithms. Many classical problems like shortest paths, minimum spanning tree and maximum matching have efficient (polynomial-time) algorithms, while many other problems like travelling sales person, graph coloring, maximum independent set and max-cut are NP-hard. It is unlikely for the NP-hard problems to have polynomial-time algorithms as it is highly believed that $P \neq NP$.

Some of the standard ways to cope with NP-hardness are: (i) approximation algorithms and more precisely, designing polynomial-time approximation schemes, (ii) heuristics which do not give any theoretical guarantee on the running time, but run reasonably well on particular practical instances, (iii) polynomial time algorithms for restricted inputs, i.e, for some special graph classes such as trees and (iv) exact exponential algorithms, a faster exact algorithm would mean increase in the size of the problem instance that can be solved in a given time.

The framework of *Parameterized Complexity*, developed by Downey and Fellows in 90s, allows to study NP-hard problems on a finer scale. The traditional computational complexity measures the running time of an algorithm with respect to the input size n (for graphs, n usually denotes the number of vertices). In parameterized complexity, apart from the input an extra parameter k is provided and the running time of the algorithm is measured in terms of both the input size n and the parameter k . The parameter k usually a positive integer indicating the size of the solution or the structure of the input.

For an NP-hard maximization (minimization) problem, computing an optimal solution in polynomial time is unlikely. However, instead of the optimal solution if we need a solution of size at least (at most) k then the problem might be tractable when k is small. The parameter solution size is usually referred to as the *natural parameter*. The

parameters which are related to the structure of the graph are called *structural parameters*. For instance, we could consider parameters like chromatic number of the graph or the maximum degree of the graph. A computational problem on graphs may be NP-hard in general but at the same time be efficiently solvable for bipartite graphs or graphs with small degree. Hence it makes sense to look at the running time in terms of n and k , where k is the parameter. When k is small if we get an efficient algorithm that leads to the definition of FPT.

A problem is called *Fixed Parameter Tractable (FPT)* with respect to a parameter k , if there is an algorithm with time complexity $O(f(k).n^c)$, where $f(k)$ is a function dependent only on the input parameter k and c is a constant. If a problem is in FPT with respect to a parameter k , we get a polynomial-time algorithm for the problem for fixed values of k . There is a related notion of XP algorithms (*Slice-wise Polynomial*), where the running time is of the form $O(f(k) \cdot n^{g(k)})$, for some functions f and g . However the notion of XP algorithms did not gain much attention compared to FPT algorithms. In this thesis also we focus on FPT algorithms.

One of the commonly studied parameters is the tree-width of the graph. Unlike maximum degree and solution size, tree-width does not have a simple definition. Tree-width measures how close the graph is to a tree. Indeed several NP-hard problems like maximum independent set, chromatic number, minimum dominating set have straightforward dynamic programming algorithms when the input graph is a tree. Tree-width generalizes this notion and defines a class of graphs that are structurally similar to trees. Many NP-hard graph problems are shown to be tractable for graphs with bounded tree-width. That is, the problems are shown to be in FPT with respect to the parameter tree-width of the graph [25]. Some of the other highly used parameters include vertex cover number – the minimum number of vertices to remove from the graph to get an empty (edgeless) graph and feedback vertex number – the minimum number of vertices to remove from the graph to get a forest (acyclic graph). Many NP-hard problems are shown to be in FPT for the parameters vertex cover number [37, 39, 52] or feedback vertex number [53].

Our main focus in this thesis will be on graph coloring/partitioning problems. The classical coloring problem is to color the vertices of the graph such that no pair of adjacent vertices are of the same color. The classical coloring problem is among the Karp's 21 NP-complete problems which he showed in 1972. Graph coloring has history starting from the famous four coloring theorem. It had several attempts and was eventually proven by Appel and Haken in 1976 using computer based case analysis. Due to its rich applications in scheduling, register allocation, time tabling, social and biological networks the graph coloring problem has been explored in all dimensions. Many variants of coloring such as list coloring [30], acyclic coloring [42], multi coloring [43], equitable coloring [65], oriented

coloring [23] and many more coloring problems has been explored in the past.

In graph partitioning problems we need to partition the vertices of the graph into two or more sets such that some condition is satisfied. The well known *minimum cut* problem is to partition the vertices into two sets such that the number of edges across the sets is minimized. Minimum cut problem is polynomial-time solvable [73]. A variant of minimum cut called $s - t$ cut which is to find a minimum cut separating two given vertices s and t . Ford and Fulkerson [35] gave a polynomial-time algorithm for the $s - t$ cut problem. The *maximum cut* problem is to partition the vertices into two sets such that the number of edges across the sets is maximized. Interestingly, the maximum cut problem is NP-hard. Apart from the above problems, many variants of graph partitioning problems such as multiway cut [26], multicut [38], minimum k -cut [40] and multimultiway cut [1] has been studied in the past.

In this thesis, we consider the NP-hard graph partitioning problems (i)MATCHING CUT, (ii) H -FREE COLORING and (iii) Happy Coloring. We study these problems from parameterized complexity perspective. The parameters we mainly consider are the solution size, tree-width and neighborhood diversity.

1.1 MATCHING CUT Problem

Consider an undirected graph G such that $|V(G)| = n$. An *edge cut* is an edge set $S \subseteq E(G)$ such that the removal of S from the graph increases the number of components in the graph. A *matching* is an edge set such that no two edges in the set share a common endpoint. A *matching cut* is an edge cut which is also a matching. The MATCHING CUT problem is the decision problem of determining whether a given graph G has a matching cut.

The MATCHING CUT problem was first introduced by Graham in [41], in the name of *decomposable graphs*. Farley and Proskurowski [31] pointed out the applications of the MATCHING CUT problem in computer networks – in studying the networks which are immune to failures of non-adjacent links.

Patrignani and Pizzonia [70] pointed out the applications of the MATCHING CUT problem in orthogonal three-dimensional graph drawing. They refer to a method of graph drawing, where one starts with a degenerate drawing where all the vertices and edges are at the same point. At each step, the vertices in the drawing are partitioned and progressively the drawing approaches the original graph. In this regard, the cut involving the non-adjacent edges (matching cut) yields a more efficient and effective performance.

1.1.1 Previous Work

The MATCHING CUT problem is NP-complete for the following graph classes:

- Graphs with maximum degree 4 (Chvátal [19], Patrignani and Pizzonia [70]).
- Bipartite graphs with one partite set has maximum degree 3 and the other partite set has maximum degree 4 (Le and Randerath [58]).
- Planar graphs with maximum degree 4 and planar graphs with girth 5 (Bonsma [11]).
- $K_{1,4}$ -free graphs with maximum degree 4 (inferred from the reduction in [19]).

The MATCHING CUT problem has polynomial-time algorithms for the following graph classes:

- Graphs with maximum degree 3 (Chvátal [19]).
- Line graphs (Moshi [67]).
- Graphs without chordless cycles of length 5 or more (Moshi [67]).
- Series parallel graphs (Patrignani and Pizzonia [70]).
- Claw-free graphs, cographs, graphs with bounded tree-width and graphs with bounded clique-width (Bonsma [11]).
- Graphs with diameter 2 (Borowiecki and Jesse-Józefczyk [12]).
- $(K_{1,4}, K_{1,4} + e)$ -free graphs (Kratsch and Le [52]).

When the graph G has degree at least 2, the MATCHING CUT problem in G is equivalent to the problem of deciding whether the line graph of G has a stable cut set. A *stable cut set* is a set $S \subseteq V(G)$ of independent vertices, such that the removal of S from the graph G increases the number of components of G . Algorithmic aspects of stable cut set of line graphs have been studied in [13, 15, 16, 17, 20, 50, 58, 59].

Recently, Kratsch and Le [52] presented a $2^{n/2}n^{O(1)}$ time algorithm for the MATCHING CUT problem using branching techniques. They also showed that the MATCHING CUT problem is tractable for graphs with bounded vertex cover.

1.1.2 Our Results

One way to show that a graph problem is FPT with respect to the parameter tree-width is to give a monadic second-order logic (MSOL) formulation for that problem. Courcelle’s theorem [21, 22] states that every problem expressible in MSOL can be solved in linear time on graphs with bounded tree-width. This leads to an algorithm with running time $f(\|\varphi\|, t) \cdot n$, where $\|\varphi\|$ is the length of the MSOL formula, t is the tree-width of the graph and n is the number of vertices of the graph. However, the function $f(\|\varphi\|, t)$ can be as bad as a tower of exponentials of height $\|\varphi\|$. We state the following excerpt from the book *Parameterized Algorithms* by Cygan et al. [25].

“Tracing the exact bound on f even for simple formulas φ is generally very hard, and depends on the actual proof of the theorem that is used. This exact bound is also likely to be much higher than optimal. For this reason, Courcelle’s theorem and its variants should be regarded primarily as classification tools, whereas designing efficient dynamic-programming routines on tree decompositions requires ‘getting your hands dirty’ and constructing the algorithm explicitly.”

Considering this, it is preferable to have explicit combinatorial algorithms, since such algorithms are more efficient and are amenable to a precise running time analysis.

The MATCHING CUT problem can be expressed using an MSOL formula [11]. MSOL along with Courcelle’s theorem yields an algorithm with time complexity $f(\|\varphi\|, t) \cdot n$. That raises the following question, asked in [52]: Can we have an algorithm where f is a single exponential function?

In this thesis, we answer the above question by giving a $2^{O(t)} \cdot n$ time algorithm for the MATCHING CUT problem, where t is the tree-width of the graph. We also show that the MATCHING CUT problem is tractable for graphs with bounded neighborhood diversity and FPT for other structural parameters.

1.2 H -Free Coloring Problems

Let G be an undirected graph. The classical q -COLORING problem is to color the vertices of the graph G using at most q colors such that no pair of adjacent vertices are of the same color. The CHROMATIC NUMBER of the graph is the minimum number of colors required for q -coloring the graph and is denoted by $\chi(G)$. The graph coloring problem has been extensively studied in various settings.

In this thesis we consider a generalization of the graph coloring problem called H -FREE q -COLORING which is to color the vertices of the graph using at most q colors such that

none of the color classes contain H as an induced subgraph. The H -FREE CHROMATIC NUMBER is the minimum number of colors required to H -free color the graph and is denoted by $\chi(H, G)$. Note that when $H = K_2$, the H -FREE q -COLORING problem is same as the traditional q -COLORING problem.

For $q \geq 3$, H -FREE q -COLORING problem is NP-complete as the q -COLORING problem is NP-complete. The 2-COLORING problem is polynomial-time solvable as it is equivalent to decide whether the graph is bipartite. The H -FREE 2-COLORING problem has been shown to be NP-complete as long as H has 3 or more vertices [2]. Rao [71] has mentioned about the MSOL formulation and linear time solvability of H -FREE COLORING problems for graphs with bounded tree-width. A variant of H -FREE COLORING problem which we call H -(SUBGRAPH)FREE q -COLORING is to color the vertices of the graph such that none of the color classes contain H as a subgraph (need not be induced) is studied in [54, 61].

1.2.1 Related Work

Graph bipartitioning (2-coloring) problems with other constraints have been explored in the past. Many variants of 2-coloring have been shown to be NP-hard. Recently, Karpiński [49] studied a problem which asks to color the vertices of the graph using 2 colors such that there is no monochromatic cycle of a fixed length. The degree bounded bipartitioning problem asks to partition the vertices of G into two sets A and B such that the maximum degree in the induced subgraphs $G[A]$ and $G[B]$ are at most a and b respectively. Xiao and Nagamochi [77] proved that this problem is NP-complete for any non-negative integers a and b except for the case $a = b = 0$, in which case the problem is equivalent to testing whether G is bipartite.

Other variants that place constraints on the degree of the vertices within the partitions have also been studied [8, 24]. Wu, Yuan and Zhao [76] showed the NP-completeness of the variant that asks to partition the vertices of the graph G into two sets such that both the induced graphs are acyclic. Farrugia [32] showed the NP-completeness of a problem called $(\mathcal{P}, \mathcal{Q})$ -coloring problem. Here, \mathcal{P} and \mathcal{Q} are any additive induced-hereditary graph properties. The problem asks to partition the vertices of G into A and B such that $G[A]$ and $G[B]$ have properties \mathcal{P} and \mathcal{Q} respectively.

1.2.2 Our Results

For a fixed q , the H -FREE q -COLORING problem can be expressed in monadic second-order logic (MSOL) [71]. The MSOL formulation together with Courcelle's theorem [21, 22] yields an algorithm with running time $f(\|\varphi\|, t) \cdot n$, where $\|\varphi\|$ is the length of the MSOL

formula, t is the tree-width of the graph and n is the number of vertices of the graph.

In this thesis, we present the following explicit combinatorial algorithms for H -free coloring problems.

- An $O(q^{O(t^r)} \cdot n)$ time algorithm for the H -FREE q -COLORING problem, where $r = |V(H)|$.
- An $O(2^{t+r \log t} \cdot n)$ time algorithm for K_r -FREE 2-COLORING, where K_r is a complete graph on r vertices.
- An $O(q^{O(t^r)} \cdot n)$ time algorithm for the H -(SUBGRAPH)FREE q -COLORING problem, where $r = |V(H)|$.
- An $O(2^{O(t^2)} \cdot n)$ time algorithm for C_4 -(SUBGRAPH)FREE 2-COLORING, where C_4 is a cycle on 4 vertices.
- An $O(2^{O(t^{r-2})} \cdot n)$ time algorithm for $\{K_r \setminus e\}$ -(SUBGRAPH)FREE 2-COLORING, where $K_r \setminus e$ is a graph obtained by removing an edge from K_r .
- An $O(2^{O((tr^2)^{r-2})} \cdot n)$ time algorithm for C_r -(SUBGRAPH)FREE 2-COLORING problem, where C_r is a cycle of length r .

For graphs with tree-width t the H -FREE CHROMATIC NUMBER (H -(SUBGRAPH)FREE CHROMATIC NUMBER) is at most $t + 1$. Hence, we have an $O(t^{O(t^r)} \cdot n \log t)$ time algorithm to compute H -FREE CHROMATIC NUMBER (H -(SUBGRAPH)FREE CHROMATIC NUMBER) for graphs with tree-width at most t . Which shows that H -FREE CHROMATIC NUMBER (H -(SUBGRAPH)FREE CHROMATIC NUMBER) is FPT for tree-width.

1.3 Happy Coloring Problems

Analyzing large networks is of fundamental importance for a constantly growing number of applications. In particular, how does one mine social networks to provide valuable insight? A basic observation concerning the structure of social networks is *homophily*, that is the principle that we tend to share characteristics with our friends. Intuitively, it seems believable our friends are similar to us in terms of their age, gender, interests, opinions, and so on. In fact, this observation is well-known in sociology (see [29, 57, 64]). For example, imagine a network of supporters in a country with a two-party system. In order to check whether there is homophily by political stance (i.e., a person tends to befriend a person with similar political beliefs), we could count the number of edges between two people of opposite beliefs. If there were no such edges, we would observe homophily in an

extreme sense. It is the characteristic of social networks that they evolve over time: links tend to be added between people that share some characteristic. But given a snapshot of the network, how extensively can homophily be present? For instance, how far can an extreme ideology spread among people some of whom are “politically neutral”?

We abstract these questions regarding the computation of homophily as follows. Consider a vertex-colored graph G . We say an edge is *happy* if its endpoints have the same color (otherwise, the edge is *unhappy*). Similarly, a vertex is *happy* if it and all its neighbors have the same color (otherwise, the vertex is *unhappy*). Equivalently, a vertex is happy when all of its incident edges are happy. Let $S \subseteq V(G)$, and let $c : S \rightarrow [k]$ be a partial vertex-coloring of G . A coloring $\tilde{c} : V(G) \rightarrow [k]$ is an *extended full coloring* of c if $\tilde{c}|_S = c$, i.e., $\tilde{c}(v) = c(v)$ for all $v \in S$. In this work, we consider the following happy coloring problems. Given a partial coloring of the vertices of the graph using k colors, the MAXIMUM HAPPY EDGES (k -MHE) problem asks to color the remaining vertices such that the number of happy edges is maximized. The MAXIMUM HAPPY VERTICES (k -MHV) problem asks to color the remaining vertices such that the number of happy vertices is maximized.

1.3.1 Previous Work

Zhang and Li [78] proved that for every $k \geq 3$, the problems k -MHE and k -MHV are NP-complete. However, when $k = 2$, they gave algorithms running in time $O(\min\{n^{2/3}m, m^{3/2}\})$ and $O(mn^7 \log n)$ for 2-MHE and 2-MHV, respectively. Towards this end, the authors used max-flow algorithms (2-MHE) and minimization of submodular functions (2-MHV). Moreover, the authors presented approximation algorithms with approximation ratios $1/2$ and $\max\{1/k, \Omega(\Delta^{-3})\}$ for k -MHE and k -MHV, respectively, where Δ is the maximum degree of the graph. Later on, Zhang, Jiang, and Li [79] gave improved algorithms with approximation ratios 0.8535 and $1/(\Delta + 1)$ for k -MHE and k -MHV, respectively.

Perhaps not surprisingly, the happy coloring problems are tightly related to cut problems. Indeed, the k -MHE problem is a generalization of the following MULTIWAY UNCUT problem [56]. In this problem, we are given an undirected graph G and a terminal set $S = \{s_1, s_2, \dots, s_k\} \subseteq V(G)$. The goal is to find a partition of $V(G)$ into classes V_1, \dots, V_k such that each class contains exactly one terminal and the total number of the edges not cut by the partition is maximized. The MULTIWAY UNCUT problem can be obtained as a special case of k -MHE, when each color is used to precolor exactly one vertex. We also mention that the complement of the MULTIWAY UNCUT problem is the MULTIWAY CUT problem that has been studied before (see e.g., [18, 26]). There are known (parameterized) algorithms for the MULTIWAY CUT problem with the size of the cut ℓ as the parameter. In this regard, the fastest known algorithm runs in $O^*(1.84^\ell)$

time [14].

1.3.2 Our Results

Apart from the results in [78] and [79], the MHE and MHV problems does not seem to be addressed for any class of graphs. In this thesis, we study the complexity of these problems for some special graph classes such as trees, bipartite graphs, split graphs and complete graphs. We also consider the weighted variants of the happy coloring problems.

We have the following results:

- We show that k -MHV and k -MHE are solvable in polynomial-time for trees. For an arbitrary k , the proposed algorithms take $O(nk \log k)$ and $O(nk)$ time respectively. We also extend our algorithms to generate all the optimal colorings of the tree. Generating each optimal coloring takes polynomial-time.
- We consider exact exponential-time algorithms for the happy coloring problems. The naive brute force runs in $k^n n^{O(1)}$ time, but we show that for every $k \geq 3$, there is an algorithm running in time $O^*(2^n)$, where n is the number of vertices in the input graph. Moreover, we prove that this is not optimal for every k by giving an even faster $O^*(1.89^n)$ -time algorithm for both 3-MHE and 3-MHV.
- We show that WEIGHTED k -MHE admits a kernel of size $k + \ell$, where ℓ is the total weight of the happy edges. The ingredients of the kernel are a polynomial-time algorithm for WEIGHTED k -MHE when the uncolored vertices induce a forest together with simple reduction rules. By combining the exact algorithm with the kernel, we obtain an algorithm running in time $2^\ell n^{O(1)}$ for k -MHE. This improves considerably upon the algorithm of Misra and Reddy [66] which runs in time $(2\ell)^{2\ell} n^{O(1)}$.
- We study the complexity of both problems for graphs with bounded tree-width and graphs with bounded neighborhood diversity. When either parameter is bounded, we show that both the problems admit polynomial-time algorithms. The result for bounded tree-width graphs was also obtained independently in [3, 66].
- We prove that for every $k \geq 3$, the problem k -MHV is NP-complete for split graphs and bipartite graphs. This extends the hardness result of Zhang and Li [78] for general graphs. Similarly, we show that k -MHE remains NP-complete for bipartite graphs. Using slightly different ideas, both problems were shown to be hard for bipartite graphs and split graphs independently of our work in [66].

- Using the result from [26] we observe that, for an arbitrary k , the k -MHE problem is NP-hard for planar graphs.
- Using the result from [27] we infer that, when the number of pre-colored vertices is bounded, the k -MHE problem can be solved in linear time for graphs with bounded branch width.

1.4 Replacement Paths Problems

Let G ($|V(G)| = n$ and $|E(G)| = m$) be an undirected graph with a weight function $w : E(G) \rightarrow \mathbb{R}_{>0}$ on the edges. Let $P_G(s, t) = \{v_0 = s, v_1, \dots, v_{l-1}, v_l = t\}$ be a shortest $s - t$ path in G . Let l denote the number of edges in $P_G(s, t)$, also denoted by $|P_G(s, t)|$. The total weight of the path $P_G(s, t)$ is denoted by $d_G(s, t)$, i.e., $d_G(s, t) = \sum_{i=1}^l w(e_i)$, where, e_i is the edge $(v_{i-1}, v_i) \in P_G(s, t)$. A shortest path tree (SPT) of G rooted at s (respectively, t) is denoted by T_s (respectively, T_t).

A *Replacement Shortest Path* (RSP) for the edge e_i (respectively, node v_i) is a shortest $s - t$ path in $G \setminus e_i$ (respectively, $G \setminus v_i$). The **EDGE REPLACEMENT PATH** problem is to compute RSP for all $e_i \in P_G(s, t)$. Similarly, the **NODE REPLACEMENT PATH** problem is to compute RSP for all $v_i \in P_G(s, t)$.

As in all existing algorithms for RSP problem, our algorithm has two phases:

1. Computing shortest path trees rooted at s and t , T_s and T_t respectively.
2. Computing RSP using T_s and T_t .

For graphs with non-negative edge weights, computing an SPT takes $O(m + n \log n)$ time, using the standard Dijkstra's algorithm [28] implemented using Fibonacci heaps [36]. However, for integer weighted graphs (RAM model) [75], planar graphs [44] and minor-closed graphs [74], $O(m + n)$ time algorithms are known. In this paper, to compute SPTs T_s and T_t (phase (i)), we use the existing algorithms. For phase (ii), we present an $O(m + l^2)$ time algorithm which is simple and easy to implement. Motivation for studying the replacement paths problem is its relevance in single link (or node) recovery protocols. Other problems which are closely related to replacement paths problem are *Most Vital Edge* problem [68], *Most Vital Node* problem [69] and *Vickrey Pricing* [45]. Often an algorithm for replacement paths problem is used as a subroutine in finding k -simple shortest paths between a pair of nodes.

1.4.1 Previous Work

For the EDGE REPLACEMENT PATH problem Malik et al. [63] and Hershberger and Suri [45] independently gave $O(T_{SPT}(G) + m + n \log n)$ time algorithms. Nardelli et al. [68] gave an $O(T_{SPT}(G) + m\alpha(m, n))$ time algorithm, where α is the inverse Ackermann function.

For the NODE REPLACEMENT PATH problem Nardelli et al. [69] gave an algorithm with time complexity $O(T_{SPT}(G) + m + n \log n)$. Kare and Saxena [48] gave an $O(T_{SPT}(G) + m\alpha(m, n))$ time algorithm.

Jay and Saxena [62] gave an $O(T_{SPT}(G) + m + d^2)$ algorithm, where d is the diameter of the graph. Their algorithm can be used to solve both the edge and the node replacement path problems. They used linear time algorithms for Range Minima Query (RMQ) [9] and integer sorting in their solution. A total of $2l$ instances, each of RMQ and integer sorting has been used (with size of each instance at most l). Recently, Lee and Lu [60] gave an $O(T_{SPT}(G) + m + n)$ time algorithm. Table 1.4.1 summarises the existing algorithms for RSP problem.

EDGE REPLACEMENT PATH Problem	
Reference	Time Complexity
Malik et al. [63] (1989)	$O(T_{SPT}(G) + m + n \log n)$
Hershberger and Suri [45] (1997)	$O(T_{SPT}(G) + m + n \log n)$
Nardelli et al. [68] (2001)	$O(T_{SPT}(G) + m\alpha(m, n))$
Jay and Saxena [62] (2013)	$O(T_{SPT}(G) + m + d^2)$
Lee and Lu [60] (2014)	$O(T_{SPT}(G) + m + n)$
This Thesis	$O(T_{SPT}(G) + m + l^2)$
NODE REPLACEMENT PATH Problem	
Reference	Time Complexity
Nardelli et al. [69] (2003)	$O(T_{SPT}(G) + m + n \log n)$
Jay and Saxena [62] (2013)	$O(T_{SPT}(G) + m + d^2)$
Kare and Saxena [48] (2014)	$O(T_{SPT}(G) + m\alpha(m, n))$
Lee and Lu [60] (2014)	$O(T_{SPT}(G) + m + n)$
This Thesis	$O(T_{SPT}(G) + m + l^2)$

Table 1.1: Summary of existing algorithms¹ for RSP problem

¹In the referenced papers, authors ignore the term $T_{SPT}(G)$, as they assume either shortest path trees are given or restriction on the input graph class for which linear time algorithms are known for SPT

1.4.2 Our Results

In this thesis, we present an $O(T_{SPT}(G) + m + l^2)$ time and $O(m + l^2)$ space algorithm. The asymptotic complexity of our algorithm matches that of [62]. However, our solution does not use RMQ and integer sorting. Our algorithm organizes the non-tree edges of the graph in a simple manner. Note that linear time algorithm for RMQ [9] and the algorithm in [60] are complex to implement. The simplicity of our algorithm makes it an ideal candidate for the RSP. In particular, for dense graphs and graphs with small diameter ($l \leq \text{diameter}(G) = O(\sqrt{m})$) our algorithm is optimal and matches with that of [60]. As observed in [62], graphs in real world data sets have small diameter, which further adds significance to our algorithm.

Chapter 2

Preliminaries

In this chapter, we give notations, terminology and concepts used in this thesis.

2.1 Graph Terminology

In this thesis, we consider simple undirected graphs. Let G be an undirected graph. We use $V(G)$ and $E(G)$ to denote the set of vertices and the set of edges of the graph respectively. We denote $|V(G)| = n$ and $|E(G)| = m$.

Neighbor Two vertices u and v are neighbors or adjacent to each other if $\{u, v\} \in E(G)$. We say that u is a neighbor of v and vice versa.

Degree Degree of a vertex u is the number of neighbors of u in the graph G and is denoted by $d_G(u)$.

Maximum Degree Maximum degree of the graph is denoted by $\Delta(G)$ and is defined as $\Delta(G) = \max_{u \in V(G)}(d_G(u))$.

Minimum Degree Minimum degree of the graph is denoted by $\delta(G)$ and is defined as $\delta(G) = \min_{u \in V(G)}(d_G(u))$.

Open Neighborhood The set of all neighbors of u is called *open neighborhood* of u and is denoted by $N_G(u)$.

Closed Neighborhood The *closed neighborhood* of u , denoted by $N_G[u]$, is defined as $N_G[u] = N_G(u) \cup \{u\}$.

Subgraph A graph H is a subgraph of the graph G if (i) $V(H) \subseteq V(G)$ and (ii) $E(H) \subseteq E(G)$ and $\{u, v\} \in E(H) \Rightarrow u, v \in V(H)$. If $V(G) = V(H)$ then the subgraph H is called a spanning subgraph.

Induced Subgraph A subgraph H of G is called an induced subgraph, if for any two vertices $u, v \in V(H)$, $\{u, v\} \in E(G) \Leftrightarrow \{u, v\} \in E(H)$. A subgraph induced by a set $S \subseteq V(G)$ is denoted by $G[S]$.

Path A path in the graph G is a subgraph P with $V(P) = \{u_0, u_1, u_2, \dots, u_\ell \mid u_i \neq u_j \forall i \neq j\}$ and $E(P) = \{\{u_i, u_{i+1}\} \mid 0 \leq i < \ell\}$.

Cycle A cycle in the graph G is a subgraph C with $V(C) = \{u_0, u_1, u_2, \dots, u_\ell \mid u_i \neq u_j \forall i \neq j\}$ and $E(P) = \{u_0, u_\ell\} \cup \{\{u_i, u_{i+1}\} \mid 0 \leq i < \ell\}$. A cycle on r vertices is denoted as C_r .

Clique A vertex subset $C \subseteq V(G)$ such that every pair of vertices in C are adjacent is called a clique.

Independent Set A vertex subset $I \subseteq V(G)$ such that every pair of vertices in I are non-adjacent is called an independent set.

Connected Graph A graph G is connected if for every two vertices $u, v \in V(G)$ there exists a path from u to v in G .

Connected Component A maximal connected subgraph of a graph is called connected component of the graph.

Edge Cut An *edge cut* is an edge set $S \subseteq E(G)$ such that the removal of S from the graph increases the number of components in the graph.

Vertex Cut A *vertex cut* is a vertex set $S \subseteq V(G)$ such that the removal of S together with all the edges incident on the vertices of S from the graph increases the number of components in the graph.

Matching A *matching* is an edge set such that no two edges in the set have a common end point.

Vertex Cover A vertex cover is a vertex set $S \subseteq V(G)$ such that for every edge $\{u, v\} \in E(G)$ either $u \in S$ or $v \in S$.

Twins Two non-adjacent (adjacent) vertices having the same open (closed) neighborhood are called *twins*.

Twin Cover A twin cover is a vertex set $S \subseteq V(G)$ such that for every edge $\{u, v\} \in E(G)$: $u \in S$ (or) $v \in S$ (or) u and v are twins.

Split Graph A graph in which the vertices can be partitioned into a clique and an independent set is called a split graph.

Distance to Split Graph The minimum size of the vertex set $S \subseteq V(G)$ such that $G[V(G) \setminus S]$ becomes a split graph is called its distance to split graph.

Proper Coloring An assignment of colors to vertices such that the adjacent vertices have different colors is called proper coloring.

Chromatic Number Minimum number of colors required to properly color the graph G is called its chromatic number and is denoted by $\chi(G)$.

Tree A connected graph without cycles is called a tree.

Forest An undirected graph where every connected component of the graph is a tree.

Subtree A connected subgraph of a tree is called subtree.

Planar Graph A graph G is said to be planar if there exists some geometric representation of G which can be drawn on a plane such that no two of its edges intersect.

Feedback Vertex Set A vertex set $S \subseteq V(G)$ such that $G[V(G) \setminus S]$ becomes a forest.

Shortest Path In an undirected graph G with weights on the edges, a shortest path $P_G(s, t)$ from s to t in G is defined as a path from s to t which minimizes the sum of the weights of the edges along the path from s to t . For unweighted graphs, the weights of edges are assumed to be 1.

Union For graphs G_1 and G_2 , the union of G_1 and G_2 is denoted by $G_1 \cup G_2$ and $V(G_1 \cup G_2) = V(G_1) \cup V(G_2)$ and $E(G_1 \cup G_2) = E(G_1) \cup E(G_2)$.

Shortest Path Tree For a distinguished vertex $s \in V(G)$ called the source vertex, and for all the vertices $u \in V(G) \setminus \{s\}$, a single source shortest path tree (SPT) in G rooted at s , denoted by T_s , is a spanning tree of G rooted at s and formed by the union of shortest paths from s to u for each $u \in V(G) \setminus \{s\}$.

Distance The distance between the two vertices u and v is the sum of the weights of the edges in the shortest path between u and v .

Diameter The diameter of the graph is the length of the maximum shortest path between all pairs of vertices.

Post-order Traversal In a rooted tree post-order traversal refers to processing or visiting of vertices such that a vertex is processed only after all its children are processed.

Pre-order Traversal In a rooted tree pre-order traversal refers to processing or visiting of vertices such that a vertex is processed before all its children are processed.

Bipartite Graph A graph whose vertices can be partitioned into two independent set I_1 and I_2 . Bipartite graphs are the graphs which do not have cycles of odd length.

Complete Graph A graph in which every pair of vertices are adjacent is called complete graph. A complete graph on r vertices is denoted as K_r .

Complete Bipartite Graph A graph whose vertices can be partitioned into two independent set I_1 and I_2 such that every vertex in I_1 is adjacent to every vertex in I_2 . A complete bipartite graph is denoted as $K_{r,s}$ where $|I_1| = r$ and $|I_2| = s$.

Diamond Graph A graph formed by deleting any one edge from K_4 is called a diamond.

Claw Graph The complete bipartite graph $K_{1,3}$ is known as claw.

Minor an undirected graph H is called a minor of G if H can be formed from G by deleting edges and vertices and by contracting edges.

Minor-closed Graphs A graph family F is called minor-closed if every minor of a graph in F is also in F . For example planar graphs are minor-closed.

Directed Acyclic Graph A directed graph without any directed cycle is called a directed acyclic graph (DAG).

In-degree In a directed graph indegree of a vertex is the number of incoming edges onto the vertex.

Out-degree In a directed graph outdegree of a vertex is the number of outgoing edges from the vertex.

H -free For a fixed graph H , a graph G is called H -free, if there is no $S \subseteq V(G)$ such that $G[S]$ is isomorphic to H .

2.2 Parameterized Complexity

A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed, finite alphabet. For an instance $(I, k) \in \Sigma^* \times \mathbb{N}$, we call k the *parameter*. The parameterized problem L is *fixed-parameter tractable* (FPT) when there is an algorithm \mathcal{A} , a computable function $f : \mathbb{N} \rightarrow \mathbb{N}$, and a constant c such that, given $(I, k) \in \Sigma^* \times \mathbb{N}$, the algorithm \mathcal{A} correctly decides whether $(I, k) \in L$ in time bounded by $f(k) \cdot |I|^c$. An equivalent way of proving a problem is FPT is by constructing a *kernel* for it. A kernel for a parameterized problem (I, k) is a polynomial-time algorithm \mathcal{B} that returns an equivalent instance (I', k') of L such that $|I'| \leq g(k)$, for some computable function $g : \mathbb{N} \rightarrow \mathbb{N}$. Here, we say two instances

are *equivalent* if the first is a YES-instance if and only if the second is a YES-instance. Given a parameterized problem, it is natural to ask whether it admits a kernel, and moreover whether that kernel is small. By small, we typically mean a polynomial kernel, or even a linear kernel (i.e., $g(k) = O(k)$).

Kernelization is often discovered through *reduction rules*. A reduction rule is a polynomial-time transformation of an instance (I, k) to another instance of the same problem (I', k') such that $|I'| < |I|$ and $k' \leq k$. A reduction rule is *safe* when the instances are equivalent. For more on parameterized complexity, we refer the interested reader to [25].

2.3 Tree-Width

A *tree decomposition* of G is a pair $(T, \{X_i, i \in I\})$, where $X_i \subseteq V(G)$ for every $i \in I$, and T is a tree with elements of I as the nodes such that:

1. For each vertex $v \in V(G)$, there is an $i \in I$ such that $v \in X_i$.
2. For each edge $\{u, v\} \in E(G)$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$.
3. For each vertex $v \in V(G)$, $T[\{i \in I \mid v \in X_i\}]$ is connected.

The width of the tree decomposition is $\max_{i \in I} (|X_i| - 1)$. The tree-width of G is the minimum width taken over all tree decompositions of G and we denote it as t . For more details on tree-width, we refer the reader to [72]. Kloks [51] introduced *nice tree decomposition*, which is a tree decomposition where every node $i \in I$ is one of the following types:

1. Leaf node: For a leaf node i , $X_i = \emptyset$.
2. Introduce Node: An introduce node i has exactly one child j and there is a vertex $v \in V(G) \setminus X_j$ such that $X_i = X_j \cup \{v\}$.
3. Forget Node: A forget node i has exactly one child j and there is a vertex $v \in V(G) \setminus X_i$ such that $X_j = X_i \cup \{v\}$.
4. Join Node: A join node i has exactly two children j_1 and j_2 such that $X_i = X_{j_1} = X_{j_2}$.

Every graph G has a nice tree decomposition with $|I| = O(n)$ nodes and width equal to the tree-width of G . Moreover, such a decomposition can be found in linear time if the tree-width is bounded [51]. A tree decomposition of the graph in Figure 2.1 is given in Figure 2.2. A nice tree decomposition of the tree decomposition in Figure 2.2 is given in Figure 2.3.

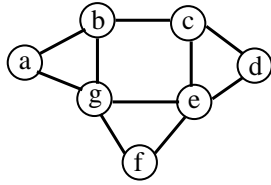


Figure 2.1: An example graph.

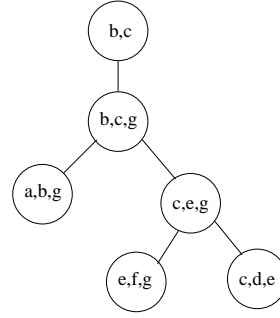


Figure 2.2: Tree decomposition of the graph G

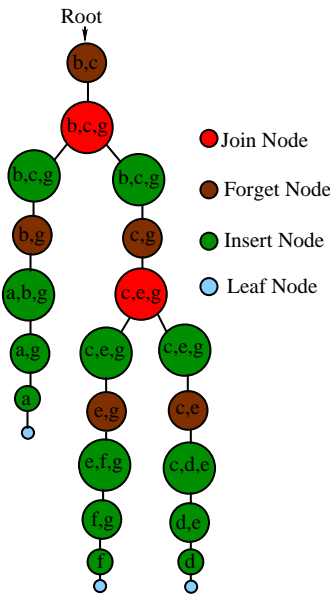


Figure 2.3: A nice tree decomposition of G

2.4 Branch-width

A branch decomposition of an undirected graph G is a pair (T, η) , where T is an unrooted binary tree with $|E(G)|$ leaves, all its internal nodes have degree 3 and η is a bijection from $E(G)$ to the leaves of T . Consider any edge $e \in E(T)$, if we remove edge e from the tree T , the T is partitioned into two subtrees and hence the set of leaves of T (edges of G) is also partitioned into two sets. Each of these sets corresponds to a subgraph in G . The width of the edge e is the number of common vertices between these two subgraphs. The width of the branch decomposition is the maximum width taken over all the edges of the tree T .

The branch-width of the graph G is the minimum width taken over all the branch decompositions of the graph G and is denoted by b . Robertson and Seymour [72] introduced the notions of tree-width and branch-width. They showed that $b(G) \leq t(G) + 1 \leq 3/2b(G)$.

2.5 MSOL and Courcelle's Theorem

Graph properties can be expressed in monadic second-order logic (MSOL). An MSOL formula is a string formed over the symbols of the MSOL using the syntactic rules of the MSOL. An MSOL formula can have four types of variables: variables for single vertex, for single edge, for subset of vertices and for subset of edges. Quantifiers are used to express graph properties. There are two types of quantifiers. The quantifier \forall is called the universal quantifier and \exists is called existential quantifier. In MSOL we can use quantifiers over variables of single vertex, single edge, vertex subset or edge subset.

Formulas of MSOL are constructed inductively from smaller formulas. Below are the smallest building blocks called atomic formulas:

1. If u is a vertex (edge) variable and X is a vertex (edge) set variable, then we can write $u \in X$. This formula is true if and only if u is an element in X .
2. If u is a vertex variable and e is an edge variable, then we can write formula $\text{inc}(u, e)$. This formula is true if and only if u is an end vertex of e .
3. For any two variables x, y of the same type, we can write formula $x = y$. This formula is true if and only if x and y are equivalent.

By using the standard boolean operators \neg , \vee , \wedge and \implies between formulas we can build larger formulas. If φ_1 and φ_2 are two formulas, φ_1 and φ_2 can be combined to form larger formula using the following operations.

$\neg\varphi_1$: $\neg\varphi_1$ is true if and only if φ_1 is false.

$\varphi_1 \wedge \varphi_2$: $\varphi_1 \wedge \varphi_2$ is true if and only if both φ_1 is true and φ_2 is true.

$\varphi_1 \vee \varphi_2$: $\varphi_1 \vee \varphi_2$ is true if and only if φ_1 is true or φ_2 is true.

$\varphi_1 \implies \varphi_2$: $\varphi_1 \implies \varphi_2$ is true if and only if φ_2 is true whenever φ_1 is true.

We can apply quantifiers over variables of single vertex ($\forall v \in V(G)/\exists v \in V(G)$), of single edge ($\forall e \in E(G)/\exists e \in E(G)$), of vertex subsets ($(\forall X \subseteq V(G)/\exists X \subseteq V(G))$) and of edge subsets ($(\forall Y \subseteq E(G)/\exists Y \subseteq E(G))$).

There are two kinds of MSOL formulas, MSO_2 which allows quantifiers over edge subsets also whereas in MSO_1 quantifiers over edge subsets is not allowed. Throughout the thesis by MSOL we mean the MSO_1 .

Below is an example MSOL formula for 3-colorability of a graph (from [25]):

$$\begin{aligned} \text{3Colorability} = & \exists X_1, X_2, X_3 \subseteq V(G) [\text{partition}(X_1, X_2, X_3) \\ & \wedge \text{indp}(X_1) \wedge \text{indp}(X_2) \wedge \text{indp}(X_3)] \end{aligned}$$

$$\begin{aligned}
\text{partition}(X_1, X_2, X_3) &= \forall v \in V[(v \in X_1 \wedge v \notin X_2 \wedge v \notin X_3) \\
&\quad \vee (v \notin X_1 \wedge v \in X_2 \wedge v \notin X_3) \\
&\quad \vee (v \notin X_1 \wedge v \notin X_2 \wedge v \in X_3)] \\
\text{indp}(X) &= \forall u, v \in X \neg \text{adj}(u, v)
\end{aligned}$$

Theorem 1 (Courcelle’s Theorem [21] from [25]). *Assume that φ is a formula of MSOL and G is a graph equipped with evaluation of all the free variables of φ . Given a tree decomposition of G of width t , there is an algorithm that verifies whether φ is satisfied in G in time $f(\|\varphi\|, t) \cdot n$ for some computable function f . Here $\|\varphi\|$ is the length of the MSOL formula and n is the number of vertices of the graph G .*

2.6 Neighborhood Diversity

The polynomial-time solvability of a problem on bounded tree-width graphs implies the existence of a polynomial-time algorithm also for other structural parameters that are polynomially upper-bounded in tree-width. For instance, one such parameter is the *vertex cover number*, i.e., the size of a smallest vertex cover that a graph has. However, graphs with bounded vertex cover number are highly restricted, and it is natural to look for less restricting parameters that generalize vertex cover (like tree-width). Another parameter generalizing vertex cover is *neighborhood diversity*, introduced by Lampis [55]. Let us first define the parameter, and then discuss its connection to both vertex cover and tree-width.

Definition 1. *In an undirected graph G , two vertices u and v have the same type if and only if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$.*

Definition 2 (Neighborhood diversity [55]). *A graph G has neighborhood diversity d if there exists a partition of $V(G)$ into d sets P_1, P_2, \dots, P_d such that all the vertices in each set have the same type. Such a partition is called a type partition. Moreover, it can be computed in linear time [55].*

Note that all the vertices in P_i for every $i \in [d]$ have the same neighborhood in G . Moreover, each P_i either forms a clique or an independent set in G .

Neighborhood diversity can be viewed as representing the simplest of dense graphs. If a graph has vertex cover number q , then the neighborhood diversity of the graph is not more than $2^q + q$ (for a proof, see [55]). Hence, graphs with bounded vertex cover number also have bounded neighborhood diversity. However, the converse is not true since complete graphs have neighborhood diversity 1. Paths and complete graphs also show that neighborhood diversity and tree-width are incomparable. In general, some NP-hard

problems (some of which remain hard for tree-width), are rendered tractable for bounded neighborhood diversity (see e.g., [33, 37, 39]).

2.7 Problem Definitions

The MATCHING CUT problem is defined as follows:

Definition 3. MATCHING CUT *problem*

(Instance) An undirected graph G .

(Question) Does the graph has a matching cut?

The H -FREE q -COLORING and H -FREE CHROMATIC NUMBER problems are defined as follows:

Definition 4. H -FREE q -COLORING

(Instance) An undirected graph G .

(Question) Can we color the vertices of G using at most q colors such that none of the color classes contain H as an induced subgraph?

Definition 5. H -FREE CHROMATIC NUMBER

(Instance) An undirected graph G .

(Output) The minimum number of colors required to color the vertices of the graph such that none of the color classes contain H as an induced subgraph.

The H -(SUBGRAPH)FREE q -COLORING and H -(SUBGRAPH)FREE CHROMATIC NUMBER problems are defined as follows:

Definition 6. H -(SUBGRAPH)FREE q -COLORING

(Instance) An undirected graph G .

(Question) Can we color the vertices of G using at most q colors such that none of the color classes contain H as a subgraph?

Definition 7. H -(SUBGRAPH)FREE CHROMATIC NUMBER

(Instance) An undirected graph G .

(Output) The minimum number of colors required to color the vertices of the graph such that none of the color classes contain H as a subgraph.

The unweighted variants of the happy coloring problems are defined as follows:

Definition 8. MAXIMUM HAPPY EDGES (MHE)

(Instance) A graph G , integers k , a vertex subset $S \subseteq V(G)$, (partial) coloring $c : S \rightarrow [k]$.

(Output) A coloring $\tilde{c} : V(G) \rightarrow [k]$ such that $\tilde{c}|_S = c$ maximizing the total number of the happy edges.

Definition 9. MAXIMUM HAPPY VERTICES (MHV)

(Instance) A graph G , integers k , a vertex subset $S \subseteq V(G)$, (partial) coloring $c : S \rightarrow [k]$.

(Output) A coloring $\tilde{c} : V(G) \rightarrow [k]$ such that $\tilde{c}|_S = c$ maximizing the total number of the happy vertices.

When k is fixed and not part of the input, we refer the MHE and MHV problems as k -MHE and k -MHV respectively. The decision versions of the unweighted variants of the happy coloring problems are defined as follows:

Definition 10. DECISION MHE (DMHE)

(Instance) A graph G , integers k and ℓ , a vertex subset $S \subseteq V(G)$, (partial) coloring $c : S \rightarrow [k]$.

(Question) Does there exist a coloring $\tilde{c} : V(G) \rightarrow [k]$ such that $\tilde{c}|_S = c$ and the number of the happy edges is at least ℓ ?

Definition 11. DECISION MHV (DMHV)

(Instance) A graph G , integers k and ℓ , a vertex subset $S \subseteq V(G)$, (partial) coloring $c : S \rightarrow [k]$.

(Question) Does there exist a coloring $\tilde{c} : V(G) \rightarrow [k]$ such that $\tilde{c}|_S = c$ and the number of the happy vertices is at least ℓ ?

The weighted variants of the happy coloring problems are defined as follows:

Definition 12. WEIGHTED MAXIMUM HAPPY EDGES (WEIGHTED MHE)

(Instance) A graph G , integers k , a vertex subset $S \subseteq V(G)$, (partial) coloring $c : S \rightarrow [k]$, and a weight function $w : E(G) \rightarrow \mathbb{N}$.

(Output) A coloring $\tilde{c} : V(G) \rightarrow [k]$ such that $\tilde{c}|_S = c$ maximizing the total weight of the happy edges.

Definition 13. WEIGHTED MAXIMUM HAPPY VERTICES (WEIGHTED MHV)

(Instance) A graph G , integers k , a vertex subset $S \subseteq V(G)$, (partial) coloring $c : S \rightarrow [k]$, and a weight function $w : V(G) \rightarrow \mathbb{N}$.

(Output) A coloring $\tilde{c} : V(G) \rightarrow [k]$ such that $\tilde{c}|_S = c$ maximizing the total weight of the happy vertices.

When k is fixed and not part of the input, we refer the WEIGHTED MHE and WEIGHTED MHV problems as WEIGHTED k -MHE and WEIGHTED k -MHV respectively. The decision versions of the weighted variants of the happy coloring problems are defined as follows:

Definition 14. WEIGHTED DMHE

(Instance) A graph G , integers k and ℓ , a vertex subset $S \subseteq V(G)$, (partial) coloring $c : S \rightarrow [k]$, and a weight function $w : E(G) \rightarrow \mathbb{N}$.

(Question) Does there exist a coloring $\tilde{c} : V(G) \rightarrow [k]$ such that $\tilde{c}|_S = c$ and the sum of the weights of the happy edges is at least ℓ ?

Definition 15. WEIGHTED DMHV

(Instance) A graph G , integers k and ℓ , a vertex subset $S \subseteq V(G)$, (partial) coloring $c : S \rightarrow [k]$, and a weight function $w : V(G) \rightarrow \mathbb{N}$.

(Question) Does there exist a coloring $\tilde{c} : V(G) \rightarrow [k]$ such that $\tilde{c}|_S = c$ and the sum of the weights of the happy vertices is at least ℓ ?

Problem definitions for replacement shortest paths problems.

Definition 16. EDGE REPLACEMENT PATH

(Instance) An undirected graph G with positive edge weights, two specified vertices s, t and shortest $s - t$ path $P_G(s, t)$ in G .

(Output) A shortest $s - t$ path in $G \setminus \{e\}$, for every edge e in $P_G(s, t)$.

Definition 17. NODE REPLACEMENT PATH

(Instance) An undirected graph G with positive edge weights, two specified vertices s, t and shortest $s - t$ path $P_G(s, t)$ in G .

(Output) A shortest $s - t$ path in $G \setminus \{v\}$, for every vertex v in $P_G(s, t)$.

Other partitioning problems which are used in the thesis are:

Definition 18. MAX WEIGHTED PARTITION

(Instance) An n -element set N , integer d , and functions $f_1, f_2, \dots, f_d : 2^N \rightarrow [-M, M]$ for some integer M .

(Output) A d -partition (S_1, S_2, \dots, S_d) of N that maximizes $f_1(S_1) + f_2(S_2) + \dots + f_d(S_d)$.

Definition 19. MULTIWAY CUT

(Instance) An undirected graph G and a terminal set $S = \{s_1, s_2, \dots, s_k\} \subseteq V(G)$.

(Output) A set of edges $C \subseteq E(G)$ with minimum cardinality whose removal disconnects all the terminals from each other.

Definition 20. MULTIWAY UNCUT

(Instance) An undirected graph G and a terminal set $S = \{s_1, s_2, \dots, s_k\} \subseteq V(G)$.

(Output) A partition $\{V_1, V_2, \dots, V_k\}$ of $V(G)$ such that each partition contains exactly one terminal and the number of edges not cut by the partition is maximized.

Definition 21. MULTI-MULTIWAY CUT

(Instance) An undirected graph G and c sets of vertices S_1, S_2, \dots, S_c .

(Output) A set of edges $C \subseteq E(G)$ with minimum cardinality whose removal disconnects every pair of vertices in each set S_i .

2.8 Other Notations

We write $f(n) = O^*(g(n))$ if $f(n) = O(g(n)n^c)$ for some constant $c > 0$, here $g(n)$ be any function on n . When there is no ambiguity, we use the simpler notations $S \setminus x$ to denote $S \setminus \{x\}$ and $S \cup x$ to denote $S \cup \{x\}$. We denote the set of all k sized subsets of the set S by $\binom{S}{k}$. Some times we use uv to denote the edge $\{u, v\}$ for convenience. We denote the set $\{1, 2, 3, \dots, r\}$ by $[r]$.

2.9 Organization

In Chapter 3 we discuss the parameterized algorithms for the MATCHING CUT problem. In Chapter 4 we discuss the parameterized algorithms for the H -FREE q -COLORING problems and its variants. In Chapter 5 we discuss results on happy coloring problems; polynomial-time algorithms, hardness results for special graph classes and parameterized algorithms. In Chapter 6 we discuss algorithms for replacement shortest path problems. Finally we give conclusions and future work in Chapter 7.

Chapter 3

Algorithms for Matching Cut Problem

The MATCHING CUT problem is a graph partitioning problem, where we need to partition the vertices into two non-empty sets A and B such that the edges across the sets induce a matching.

The MATCHING CUT problem can be expressed using a monadic second-order logic (MSOL) formula [11]. The MSOL formulation, together with Courcelle's theorem implies linear time solvability on graphs with bounded tree-width. This approach yields an algorithm with running time $f(\|\varphi\|, t) \cdot n$. Where $\|\varphi\|$ is the length of the MSOL formula and t is the tree-width of the graph. However, the function $f(\|\varphi\|, t)$ can be as bad as a tower of exponentials of height $\|\varphi\|$. That raises the following question, asked in [52]: Can we have an algorithm where f is a single exponential function?

In this thesis, we answer the above question by giving a $2^{O(t)} \cdot n$ time explicit combinatorial algorithm for the MATCHING CUT problem, where t is the tree-width of the graph. We also show that the MATCHING CUT problem is tractable for graphs with bounded neighborhood diversity and other structural parameters.

3.1 Graphs with Bounded Tree-width

In this section, we present an $O(2^{O(t)} \cdot n)$ time algorithm for the MATCHING CUT problem. The algorithm we present is based on dynamic programming technique on the nice tree decomposition. We use the following notations in the algorithm.

- i : A node in the tree decomposition.
- X_i : The set of vertices associated with node i . The X_i s will sometimes be referred as bags.

- $G[X_i]$: Subgraph induced by X_i .
- T_i : The sub-tree rooted at node i of the tree decomposition. This includes node i and all its descendants.
- $G[T_i]$: Subgraph induced by the vertices in node i and all its descendants.

Let $\Psi = (A_1, A_2, A_3, B_1, B_2, B_3)$ be a partition of X_i , we say that the partition Ψ is **legal** at node i if it satisfies the following conditions (\star):

1. Every vertex of A_1 (respectively B_1) has exactly one neighbor in B_1 (resp. A_1) and no neighbors in $B_2 \cup B_3$ (resp. $A_2 \cup A_3$).
2. Every vertex of $A_2 \cup A_3$ (resp. $B_2 \cup B_3$) has no neighbors in any of the B_i 's (resp. A_i 's).

We say that a legal partition ψ is **valid** for the node i if there exists a matching cut (A, B) of $G[T_i]$ such that the following conditions ($\star\star$) hold:

1. The A_i 's are contained in A and the B_i 's are contained in B .
2. Every vertex of A_1 (resp. B_1) has a matching cut neighbor in B_1 (resp. A_1).
3. Every vertex of $A_2 \cup B_2$ has a matching cut neighbor in $G[T_i] \setminus X_i$.
4. The vertices of $A_3 \cup B_3$ are not part of the cut-edges, i.e. every vertex of A_3 (resp. B_3) has no neighbor in B (resp. A).

A matching cut is empty if there are no edges in cut. We say that a valid partition Ψ of X_i is *locally empty* in $G[T_i]$, if every matching cut of $G[T_i]$ extending Ψ (i.e. satisfying $\star\star$) is empty. Note that, a necessary condition for Ψ to be locally empty is: $A_1 \cup A_2 \cup B_1 \cup B_2 = \emptyset$.

We define $M_i[\Psi]$ to be $+1$ if Ψ is valid for the node X_i and not locally empty, 0 if it is valid and locally empty, and -1 otherwise. Now, we explain how to compute $M_i[\Psi]$ for each partition Ψ at the nodes of the nice tree decomposition.

Leaf node: For a leaf node i , $X_i = \emptyset$. We have $\Psi = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$ and $M_i[\Psi] = 0$. This step can be executed in constant time.

Introduce node: Let j be the only child of the node i . Suppose that $v \in X_i$ is the new node present in X_i , $v \notin X_j$. Let $\Psi = (A_1, A_2, A_3, B_1, B_2, B_3)$ be a partition of X_i . If Ψ is not legal, we straightaway set $M_i[\Psi]$ to -1 . Otherwise, we use the below procedure to compute $M_i[\Psi]$ for $v \in A_i$, and analogously for $v \in B_i$.

Case 1: $v \in A_1$. $M_i[\Psi] = +1$, if there exists a unique $x \in B_1$, such that, $(v, x) \in E(G)$ and $M_j[\Psi'] \geq 0$ for $\Psi' = (A_1 \setminus v, A_2, A_3, B_1 \setminus x, B_2, B_3 \cup x)$. Otherwise $M_i[\Psi] = -1$. Note that, $M_i[\Psi]$ can not be 0, as $v \in A_1$ brings an edge into the cut if it is valid.

Case 2: $v \in A_2$. This case is not valid as v does not have any neighbor in $V(T_i) \setminus X_i$ (it is the property of the nice tree decomposition).

Case 3 $v \in A_3$. $M_i[\Psi] = M_j[\Psi']$ where $\Psi' = (A_1, A_2, A_3 \setminus v, B_1, B_2, B_3)$.

The total number of possible Ψ 's for X_i is 6^{t+1} . For each Ψ , the above cases can be executed in polynomial-time. Hence, the total time complexity at the introduce node is $O^*(6^t)$.

Forget node: Let j be the only child of the node i . Suppose, $v \in X_j$ is the node missing in X_i , $v \notin X_i$. Let $\Psi = (A_1, A_2, A_3, B_1, B_2, B_3)$ be a partition of X_i . If Ψ is not legal, we straightaway set $M_i[\Psi]$ to -1 .

Otherwise, $M_i[\Psi] = \max_{k=1}^6 \{\delta_k\}$, where δ_k is computed as follows: If Ψ is valid, it should be possible to add v to one of the six sets to get a valid partition at node j .

Case 1: v is in the first set at the node j . If there is a unique $x \in B_2$ such that $(v, x) \in E(G)$ then $\delta_1 = M_j[\Psi']$ where $\Psi' = (A_1 \cup v, A_2, A_3, B_1 \cup x, B_2 \setminus x, B_3)$. If no such x exists, then δ_1 is set to -1 .

Case 2: v is in the second set at the node j .

Let $\Psi' = (A_1, A_2 \cup v, A_3, B_1, B_2, B_3)$ and $\delta_2 = M_j[\Psi']$.

Case 3: v is in the third set at the node j .

Let $\Psi' = (A_1, A_2, A_3 \cup v, B_1, B_2, B_3)$ and $\delta_3 = M_j[\Psi']$.

The values δ_4 , δ_5 and δ_6 are computed analogously. The total number of possible Ψ 's for X_i is 6^t . For each Ψ , the above cases can be executed in polynomial-time. Hence, the total time complexity at the forget node is $O^*(6^t)$.

Join node: Let j_1 and j_2 be the children of the node i . $X_i = X_{j_1} = X_{j_2}$ and $V(T_{j_1}) \cap V(T_{j_2}) = X_i$. There are no edges between $V(T_{j_1}) \setminus X_i$ and $V(T_{j_2}) \setminus X_i$. Let $\Psi = (A_1, A_2, A_3, B_1, B_2, B_3)$ be a partition of X_i . For $X \subseteq A_2$ and $Y \subseteq B_2$ let $\Psi_1 = (A_1, X, A_3 \cup \{A_2 \setminus X\}, B_1, Y, B_3 \cup \{B_2 \setminus Y\})$ and $\Psi_2 = (A_1, A_2 \setminus X, A_3 \cup X, B_1, B_2 \setminus Y, B_3 \cup Y)$.

$$M_i[\Psi] = \begin{cases} +1, & \text{If } \exists X \subseteq A_2 \text{ and } Y \subseteq B_2 \text{ such that } M_{j_1}[\Psi_1] + M_{j_2}[\Psi_2] \geq 1; \\ 0, & \text{If } \Psi \text{ is locally empty, (i.e } M_{j_1}[\Psi] = 0 \text{ and } M_{j_2}[\Psi] = 0); \\ -1, & \text{Otherwise} \end{cases}$$

The total number of possible Ψ 's for X_i is 6^{t+1} . For each Ψ , we need to check 2^{t+1} different Ψ_1 and Ψ_2 . The total time complexity at the join node is $O^*(12^t)$.

At each node i , let $\Delta_i = \max_{\Psi} \{M_i[\Psi]\}$. If $\Delta_i = +1$, then $G[T_i]$ has a valid non-empty matching cut. If r is the root of the nice tree decomposition, the graph G has a matching cut if $\Delta_r = +1$. By induction and the correctness of $M_i[\Psi]$ values, we can conclude the correctness of the algorithm. The total time complexity of the algorithm is $O^*(12^t) = O^*(2^{O(t)})$.

Theorem 2. *There is an algorithm with running time $O^*(2^{O(t)})$ that solves the MATCHING CUT problem, where t is the tree-width of the graph.*

3.2 Graphs with Bounded Neighborhood Diversity

Let d be the neighborhood diversity of the graph and P_1, P_2, \dots, P_d be the type partitioning of the graph. Due to the property of type partitioning, each P_i forms either a clique or an independent set in G .

Here, we show that the MATCHING CUT problem is tractable for graphs with bounded neighborhood diversity. We describe an algorithm with time complexity $O^*(2^{2d})$, where d is the neighborhood diversity of the graph.

We start with a graph G , and its type partitioning P_1, P_2, \dots, P_d . We label the vertices of G (using the type partitioning) such that vertices having the same label should be entirely on one side of the cut. We assume that the graph is connected and so is the type partitioning graph. We say that a set P_i is an I -set if P_i induces an independent set. Similarly, we say that a set P_i is a C -set if P_i induces a clique. The size of a set P_i is the number of vertices in the set P_i .

Observe that a clique K_c with $c \geq 3$ and $K_{r,s}$ with $r \geq 2$ and $s \geq 3$ do not have a matching cut. It means that all the vertices of these graphs should be entirely on one side of the cut. Consider a partition P_i , vertices of P_i are labeled according to the following rules in order:

- If P_i is a C -set with size ≥ 2 , vertices in the set P_i and all the vertices in its neighboring sets get the same label.
- If P_i is an I -set with size ≥ 3 and is adjacent to an I -set with size ≥ 2 , then the vertices in both the sets get the same label.
- If P_i is an I -set with size ≥ 3 and is adjacent to two or more sets of size ≥ 1 , then vertices in all these sets get the same label.

- If P_i is an I -set with size ≥ 3 and has only one adjacent set of size 1, then G has a matching cut.
- If P_i is an I -set with size 2 and is adjacent to an I -set of size 2 and a set of size 1, then vertices in all these sets get the same label.
- If P_i is an I -set with size 2 and is adjacent to only one I -set of size 2, in these two sets, each vertex will get different label.
- If P_i is an I -set with size 2 and is adjacent to two sets of size 1, in these three sets, each vertex will get different label.
- If P_i is an I -set with size 2 and is adjacent to a set of size 1, then G has a matching cut.
- All the remaining sets of size 1 will get different labels.

If we apply the above rules, either we conclude that G has a matching cut, or for each set we use at most 2 labels, hence we can state the following:

Lemma 3. *The number of labels required is at most $2d$.*

The vertices of each label should entirely be in the same set of the matching cut. Hence, there are 2^{2d} possible label combinations. Thus we have the following:

Theorem 4. *There is an algorithm with running time $O^*(2^{2d})$ that solves the MATCHING CUT problem, where d is the neighbourhood diversity of the graph.*

3.3 Other Structural Parameters

For graphs with bounded feedback vertex number, the tree-width is also bounded. As the MATCHING CUT problem is in FPT for tree-width, it is also in FPT for feedback vertex number. Kratsch and Le [52] showed that the MATCHING CUT problem is in FPT for the size of the vertex cover. We use the techniques used in [52] to show that the MATCHING CUT problem is in FPT for the parameters *twin cover* and the *distance to split graphs*.

Lemma 5 (stated as Lemma 3 in [52]). *Let I be an independent set and let $U = V(G) \setminus I$. Given a partition (X, Y) of U , it can be decided in $O(n^2)$ time if the graph has a matching cut (A, B) such that $X \subseteq A$ and $Y \subseteq B$.*

Two non-adjacent (adjacent) vertices having the same open (closed) neighborhood are called *twins*. A *twin cover* is a vertex set S such that for each edge $\{u, v\} \in E(G)$, either $u \in S$ or $v \in S$ or u and v are twins. Note that, for a twin cover $S \subseteq V(G)$, $G[V(G) \setminus S]$ is a collection of disjoint cliques.

Lemma 6. *Let $S \subseteq V(G)$ be a twin cover of G . Given a partition (X, Y) of S , it can be decided in $O(n^2)$ time if the graph has a matching cut (A, B) such that $X \subseteq A$ and $Y \subseteq B$.*

Proof. Clearly, $V(G) \setminus S$ induces a collection of disjoint cliques. Consider a maximal clique C on two or more vertices in $V(G) \setminus S$. Let u, v be any two vertices of the clique C . Clearly, u and v are twins. If u and v has a common neighbor in both X and Y , then the graph has no matching cut such that $X \subseteq A$ and $Y \subseteq B$. Hence, without loss of generality we can assume that u and v have common neighbors only in X . Let $X' = X \cup V(C)$. Clearly, $V(G) \setminus (S \cup V(C))$ is an independent set. Using Lemma 5, we can decide in $O(n^2)$ time if the graph has a matching cut (A, B) such that $X' \subseteq A$ and $Y \subseteq B$. \square

Let S be a twin cover of the graph. By guessing a partition (X, Y) of S , we can check in $O(n^2)$ time if G has a matching cut (A, B) such that $X \subseteq A$ and $Y \subseteq B$. Hence we can state the following theorem.

Theorem 7. *There is an algorithm with running time $O^*(2^{|S|})$ to solve the MATCHING CUT problem, where S is the twin cover of the graph.*

Lemma 8. *Let G be a graph with vertex set $V(G)$, if $S \subseteq V(G)$ be such that $G[V(G) \setminus S]$ is a split graph. Given a partition (X, Y) of S , it can be decided in $O(n^2)$ time whether the graph G has a matching cut (A, B) such that $X \subseteq A$ and $Y \subseteq B$.*

Proof. Let $V(G) \setminus S = C \cup I$ be the vertex set of the split graph, where C is a clique and I is an independent set. If $|C| = 1$ or $|C| \geq 3$, then let $X' = X \cup V(C)$ and $Y' = Y \cup V(C)$. Clearly, $V(G) \setminus (S \cup V(C))$ is an independent set. Hence, G has matching cut (A, B) such that $X \subseteq A$ and $Y \subseteq B$ if and only if G has a matching cut such that either $X' \subseteq A$ and $Y \subseteq B$ or $X \subseteq A$ and $Y' \subseteq B$. Both these instances can be solved in $O(n^2)$ time using Lemma 5. If $|C| = 2$, depending on whether the vertices of C go to X or Y , we solve four instances of Lemma 5 to check whether the graph has a matching cut (A, B) such that $X \subseteq A$ and $Y \subseteq B$. Therefore the time complexity is $O(n^2)$. \square

Similar to Theorem 7, we can state the following theorem.

Theorem 9. *There is an algorithm with running time $O^*(2^{|S|})$ to solve the MATCHING CUT problem, where $S \subseteq V(G)$ such that $G[V(G) \setminus S]$ is a split graph.*

Chapter 4

Algorithms for H -Free Coloring Problems

Let G be an undirected graph. The classical q -COLORING problem asks to color the vertices of the graph using at most q colors such that no pair of adjacent vertices are of the same color. The CHROMATIC NUMBER of the graph is the minimum number of colors required for properly coloring the graph and is denoted by $\chi(G)$. The graph coloring problem has been extensively studied in various settings.

In this thesis we consider a generalization of the graph coloring problem called H -FREE q -COLORING which asks to color the vertices of the graph using at most q colors such that none of the color classes contain H as an induced subgraph. The H -FREE CHROMATIC NUMBER is the minimum number of colors required to H -free color the graph and is denoted by $\chi(H, G)$. Note that when $H = K_2$, the H -FREE q -COLORING problem is same as the traditional q -COLORING problem.

For $q \geq 3$, H -FREE q -COLORING problem is NP-complete as the q -COLORING problem is NP-complete. The 2-COLORING problem is polynomial-time solvable. The H -FREE 2-COLORING problem has been shown to be NP-complete as long as H has 3 or more vertices [2]. A variant of H -FREE COLORING problem which we call H -(SUBGRAPH)FREE q -COLORING which asks to color the vertices of the graph such that none of the color classes contain H as a subgraph (need not be induced) is studied in [54, 61].

For a fixed q , the H -FREE q -COLORING problem can be expressed in monadic second-order logic (MSOL) [71]. The MSOL formulation together with Courcelle's theorem [21, 22] implies linear time solvability on graphs with bounded tree-width. This approach yields algorithm with running time $f(\|\varphi\|, t) \cdot n$, where $\|\varphi\|$ is the length of the MSOL formula, t is the tree-width of the graph and n is the number of vertices of the graph. The dependency of $f(\|\varphi\|, t)$ on $\|\varphi\|$ can be as bad as a tower of exponentials.

In this thesis we present the following explicit combinatorial algorithm for H -free

coloring problems.

- An $O(2^{t+r \log t} \cdot n)$ time algorithm for K_r -FREE 2-COLORING, where K_r is a complete graph on r vertices.
- An $O(q^{O(t^r)} \cdot n)$ time algorithm for the H -FREE q -COLORING problem, where $r = |V(H)|$.
- An $O(2^{O(t^2)} \cdot n)$ time algorithm for C_4 -(SUBGRAPH)FREE 2-COLORING, where C_4 is a cycle on 4 vertices.
- An $O(2^{O(t^{r-2})} \cdot n)$ time algorithm for $\{K_r \setminus e\}$ -(SUBGRAPH)FREE 2-COLORING, where $K_r \setminus e$ is a graph obtained by removing an edge from K_r .
- An $O(2^{O((tr^2)^{r-2})} \cdot n)$ time algorithm for C_r -(SUBGRAPH)FREE 2-COLORING problem, where C_r is a cycle of length r .
- An $O(q^{O(t^r)} \cdot n)$ time algorithm for the H -(SUBGRAPH)FREE q -COLORING problem, where $r = |V(H)|$.

For graphs with tree-width t the H -FREE CHROMATIC NUMBER (H -(SUBGRAPH)FREE CHROMATIC NUMBER) is at most $t + 1$. Hence, we have an $O(t^{O(t^r)} n \log t)$ time algorithm to compute H -FREE CHROMATIC NUMBER (H -(SUBGRAPH)FREE CHROMATIC NUMBER) for graphs with tree-width t . This implies that H -FREE CHROMATIC NUMBER (H -(SUBGRAPH)FREE CHROMATIC NUMBER) problem is FPT with respect to the parameter tree-width.

4.1 Overview of the Techniques Used

In the rest of the chapter, we assume that the nice tree decomposition is given. Let i be a node in the nice tree decomposition, X_i is the bag of vertices associated with the node i . Let T_i be the subtree rooted at the node i , $G[T_i]$ denote the graph induced by all the vertices in T_i .

We use dynamic programming on the nice tree decomposition to solve the problems. We process the nodes of nice tree decomposition according to its post order traversal. We say that a partition (A, B) of G is a *valid* partition if neither $G[A]$ nor $G[B]$ have H as an induced subgraph. At each node i , we check each bipartition (A_i, B_i) of the bag X_i to see if (A_i, B_i) leads to a valid partition in the graph $G[T_i]$. For each partition, we also keep some extra information that will help us to detect if the partition leads to an invalid partition at some ancestral (parent) node. We have four types of nodes in the

tree decomposition – leaf, introduce, forget and join nodes. In the algorithm, we explain the procedure for updating the information at each of these above types of nodes and consequently, to certify whether a partition is valid or not.

4.2 H -Free Coloring

Before discussing the algorithm for the general H -FREE q -COLORING problem, we discuss algorithms for K_r -FREE 2-COLORING and H -FREE 2-COLORING problems. Finally we discuss the algorithm for the general H -FREE q -COLORING problem.

4.2.1 K_r -FREE 2-COLORING

Let $\Psi = (A_i, B_i)$ be a partition of a bag X_i . We set $M_i[\Psi]$ to 1 if there exist a partition (A, B) of $V[T_i]$ such that $A_i \subseteq A$, $B_i \subseteq B$ and both $G[A]$ and $G[B]$ are K_r -free. Otherwise, $M_i[\Psi]$ is set to 0.

Leaf node: For a leaf node $\Psi = (\emptyset, \emptyset)$ and $M_i[\Psi] = 1$.

Introduce node: Let j be the only child of the node i . Suppose, $v \in X_i$ is the new vertex present in X_i , $v \notin X_j$. Let $\Psi = (A_i, B_i)$ be a partition of X_i . If $G[A_i]$ or $G[B_i]$ has K_r as a subgraph, we set $M_i[\Psi]$ to 0. Otherwise, we use the following cases to compute $M_i[\Psi]$ value. Since v cannot have forgotten neighbors, it can form a K_r only within the bag X_i .

Case 1: $v \in A_i$, $M_i[\Psi] = M_j[\Psi']$, where $\Psi' = (A_i \setminus v, B_i)$.

Case 2: $v \in B_i$, $M_i[\Psi] = M_j[\Psi']$, where $\Psi' = (A_i, B_i \setminus v)$.

Forget node: Let j be the only child of the node i . Suppose, $v \in X_j$ is the vertex missing in X_i , $v \notin X_i$. Let $\Psi = (A_i, B_i)$ be a partition of X_i . If $G[A_i]$ or $G[B_i]$ has K_r as a subgraph, we set $M_i[\Psi]$ to 0. Otherwise, $M_i[\Psi] = \max\{M_j[\Psi'], M_j[\Psi'']\}$, where, $\Psi' = (A_i \cup v, B_i)$ and $\Psi'' = (A_i, B_i \cup v)$.

Join node: Let j_1 and j_2 be the children of the node i . $X_i = X_{j_1} = X_{j_2}$ and $V(T_{j_1}) \cap V(T_{j_2}) = X_i$. Let $\Psi = (A_i, B_i)$ be a partition of X_i . If $G[A_i]$ or $G[B_i]$ has K_r as a subgraph, we set $M_i[\Psi]$ to 0. Otherwise, we use the following expression to compute $M_i[\Psi]$ value. Since there are no edges between $V(T_{j_1}) \setminus X_i$ and $V(T_{j_2}) \setminus X_i$, a K_r cannot contain forgotten vertices from both T_{j_1} and T_{j_2} .

$$M_i[\Psi] = \begin{cases} 1, & \text{If } M_{j_1}[\Psi] = 1 \text{ and } M_{j_2}[\Psi] = 1. \\ 0, & \text{Otherwise.} \end{cases}$$

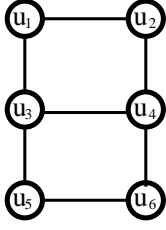


Figure 4.1: An example graph H .

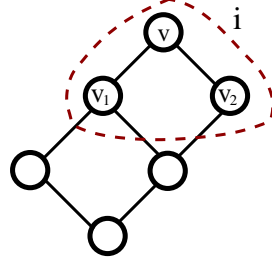


Figure 4.2: Forming H at an introduce node. Sequence $s = (v, v_2, v_1, \text{FG}, \text{FG}, \text{FG})$.

Correctness of the algorithm follows from the correctness of $M_i[\Psi]$ values, which can be proved using bottom up induction on nice tree decomposition. G has a valid bipartitioning if there exists a Ψ such that $M_r[\Psi] = 1$, where r is the root node of the nice tree decomposition. The total time complexity of the algorithm is $O(2^{t+r}n) = O^*(2^{t+r \log t})$. With this we state the following theorem.

Theorem 10. *There is an $O(2^{t+r \log t} \cdot n)$ time algorithm that solves the K_r -FREE 2-COLORING problem, on graphs with tree-width at most t .*

4.2.2 H -FREE 2-COLORING

Let X_i be a bag at node i of the nice tree decomposition. Let (A_i, B_i) be a partition of X_i . We can easily check if $G[A_i]$ or $G[B_i]$ has H as an induced subgraph. Otherwise, we need to see if there is a partition (A, B) of $V(T_i)$ such that $A_i \subseteq A$, $B_i \subseteq B$ and both $G[A]$ and $G[B]$ do not have H as an induced subgraph. If there is such a partition (A, B) , then $G[A]$ and $G[B]$ may have an induced subgraph H' , an induced subgraph of H which can lead to H at some ancestral node (introduce node or join node) of the nice tree decomposition (See Figures 4.2 and 4.3).

We perform dynamic programming over the nice tree decomposition. At each node i we guess a partition (A_i, B_i) of X_i and possible induced subgraphs of H that are part of A and B respectively. We check if such a partition is possible. Below we explain the algorithm in detail.

Let the vertices of the graph H are labeled as $u_1, u_2, u_3, \dots, u_r$. Let (A_i, B_i) be a partition of vertices in the bag X_i . Let (A, B) be a partition of $V(T_i)$ such that $A \supseteq A_i$ and $B \supseteq B_i$. We define Γ_{A_i} as follows:

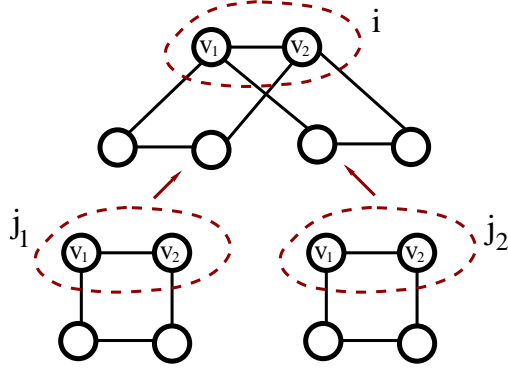


Figure 4.3: Forming H at join node. Sequences at node j_1 $s' = (\text{DC}, \text{DC}, v_1, v_2, \text{FG}, \text{FG})$, at node j_2 $s'' = (\text{FG}, \text{FG}, v_1, v_2, \text{DC}, \text{DC})$ gives a sequence $s = (\text{FG}, \text{FG}, v_1, v_2, \text{FG}, \text{FG})$ at node i . Vertices outside the dashed lines are forgotten vertices.

$$\begin{aligned}
S_{A_i} &= \{(w_1, w_2, w_3, \dots, w_r) \mid w_\ell \in \{A_i \cup \{\text{FG}, \text{DC}\}\}, \\
&\quad \forall \ell_1 \neq \ell_2, w_{\ell_1} = w_{\ell_2} \implies w_{\ell_1} \in \{\text{FG}, \text{DC}\}\}. \\
I_{A_i} &= \{s = (w_1, w_2, w_3, \dots, w_r) \in S_{A_i} \mid \text{there exists } \ell_1 \neq \ell_2 \\
&\quad \text{such that } w_{\ell_1} = \text{FG}, w_{\ell_2} = \text{DC} \text{ and } \{u_{\ell_1}, u_{\ell_2}\} \in E(H)\} \\
\Gamma_{A_i} &= S_{A_i} \setminus I_{A_i}
\end{aligned}$$

Here FG represents a vertex in $A \setminus A_i$, the forgotten vertices in A and DC stands for don't care. That is we don't care if the corresponding vertex is part of the subgraph or not. Similarly, we can define Γ_{B_i} with respect to the sets B_i and B .

A sequence in S_{A_i} corresponds to an induced subgraph H' of H in A as follows:

1. If $w_\ell = \text{FG}$ then u_ℓ is part of $A \setminus A_i$, the forgotten vertices in A .
2. If $w_\ell = \text{DC}$ then u_ℓ need not be part of the subgraph H' .
3. If $w_\ell \in A_i$ then the vertex w_ℓ corresponds to the vertex u_ℓ of H' .

Γ_{A_i} is the set of sequences that can become H in future at some ancestral (insert/join) node of the tree decomposition. Note that the sequences I_{A_i} are excluded from Γ_{A_i} because a forgot vertex cannot have an edge to a vertex which will come in future at some ancestral node (insert or join nodes).

Definition 22 (Induced Subgraph Legal Sequence in Γ_{A_i} with respect to A). *A sequence $s = (w_1, w_2, w_3, \dots, w_r) \in \Gamma_{A_i}$ is legal if the sequence s corresponds to induced subgraph H' of H within A as follows.*

Let $FV(s) = \{\ell | w_\ell = FG\}$, $DC(s) = \{\ell | w_\ell = DC\}$ and $VI(s) = [r] \setminus \{FV(s) \cup DC(s)\}$. Let H' be the induced subgraph of H formed by u_ℓ , $\ell \in \{VI(s) \cup FV(s)\}$. That is $H' = H[\{u_\ell | \ell \in VI(s) \cup FV(s)\}]$.

If there exist $|FV(s)|$ distinct vertices $z_\ell \in A \setminus A_i$ corresponding to each index in $FV(s)$ such that H' is isomorphic to $G[\{w_\ell | \ell \in VI(s)\} \cup \{z_\ell | \ell \in FV(s)\}]$, then s is legal. Otherwise, the sequence is illegal.

Similarly, we define legal/illegal sequences in Γ_{B_i} with respect to B .

Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple. Here, (A_i, B_i) is a partition of X_i , $P_i \subseteq \Gamma_{A_i}$ and $Q_i \subseteq \Gamma_{B_i}$.

We define $M_i[\Psi]$ to be 1 if there is a partition (A, B) of $V(T_i)$ such that:

1. $A_i \subseteq A$ and $B_i \subseteq B$.
2. Every sequence in P_i is legal with respect to A .
3. Every sequence in Q_i is legal with respect to B .
4. Every sequence in $\Gamma_{A_i} \setminus P_i$ is illegal with respect to A .
5. Every sequence in $\Gamma_{B_i} \setminus Q_i$ is illegal with respect to B .
6. Neither $G[A]$ nor $G[B]$ contains H as an induced subgraph.

Otherwise $M_i[\Psi]$ is set to 0.

We call a 4-tuple Ψ as invalid if one of the following conditions occur. If Ψ is invalid we set $M_i[\Psi]$ to 0.

1. There exists a sequence $s \in P_i$ such that s does not contain DC.
2. There exists a sequence $s \in Q_i$ such that s does not contain DC.

Now we explain how to compute $M_i[\Psi]$ values at the leaf, introduce, forgot and join nodes of the nice tree decomposition.

Leaf node: Let i be a leaf node, $X_i = \emptyset$, for $\Psi = (A_i, B_i, P_i, Q_i)$, we have $M_i[\Psi] = 1$. Here $A_i = B_i = \emptyset$, $P_i \subseteq \{([DC]^r)\}$ and $Q_i \subseteq \{([DC]^r)\}$.

Introduce node: Let i be an introduce node and j be the child node of i . Let $\{v\} = X_i \setminus X_j$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple at node i . If Ψ is invalid we set $M_i[\Psi] = 0$. Otherwise depending on whether $v \in A_i$ or $v \in B_i$ we have two cases. We discuss only the case $v \in A_i$, the case $v \in B_i$ can be analogously defined.

$v \in A_i$: We set $M_i[\Psi] = 0$, if there exists an illegal sequence s (in P_i) containing v or if there exists a trivial legal sequence s containing v but s is not in P_i .

That is, we set $M_i[\Psi] = 0$ in one of the following (\star) conditions occurs:

[\star Conditions]

1. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} \in A_i$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H)$ but $\{v, w_{\ell_2}\} \notin E(G)$.
2. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} \in A_i$, $\{u_{\ell_1}, u_{\ell_2}\} \notin E(H)$ but $\{v, w_{\ell_2}\} \in E(G)$.
3. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} = \text{FG}$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H)$.
4. Let $s = (w_1, w_2, w_3, \dots, w_r) \in \Gamma_{A_i} \setminus P_i$. There exists ℓ_1 such that $w_{\ell_1} = v$ and for all $\ell_2 \neq \ell_1$ $w_{\ell_2} \in A_i \cup \{\text{DC}\}$. For all $\ell_1 \neq \ell_2$ $w_{\ell_1}, w_{\ell_2} \in A_i$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H) \iff \{w_{\ell_1}, w_{\ell_2}\} \in E(G)$.

Otherwise we set $M_i[\Psi] = M_j[\Psi']$, where $\Psi' = (A_i \setminus v, B_i, P_j, Q_i)$. Here P_j is computed as follows:

Definition 23. $\text{Rep}_{\text{DC}}(s, v) = s'$, sequence s' obtained by replacing v (if present) with DC in s .

Note that, $\text{Rep}_{\text{DC}}(s, v) = s$, if v not present in s .

$$P_j = \cup_{s \in P_i} \{\text{Rep}_{\text{DC}}(s, v)\}.$$

Forget node: Let i be a forget node and j be the only child of node i . Let $\{v\} = X_j \setminus X_i$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple at node i . If Ψ is invalid we set $M_i[\Psi] = 0$. Otherwise, we set $M_i[\Psi] = \max\{\delta_1, \delta_2\}$ where δ_1 and δ_2 are computed as follows:

Computing δ_1 : Set $A_j = A_i \cup \{v\}$. As v is the extra vertex in A_j , there could be many possible P_j at node j .

Definition 24. $\text{Rep}_{\text{FG}}(s, v) = s'$, sequence s' obtained by replacing v (if present) with FG in s .

Note that, if s does not contain the vertex v then $\text{Rep}_{\text{FG}}(s, v) = s$.

We also extend the definition of Rep_{FG} to a set of sequences as follows:

$$\text{Rep}_{\text{FG}}(S, v) = \cup_{s \in S} \{\text{Rep}_{\text{FG}}(s, v)\}.$$

Note that, if s is a legal sequence at the node j with respect to A , then $\text{Rep}_{\text{FG}}(s, v)$ is also a legal sequence at node i with respect to A .

$$\delta_1 = \max_{\substack{P_j \subseteq \Gamma_{A_j} \\ \text{Rep}_{\text{FG}}(P_j, v) = P_i}} \{M_j[(A_j, B_i, P_j, Q_i)]\}$$

Computing δ_2 : $B_j = B_i \cup v$. It is analogous to computing δ_1 but we process on B .

Join node: Let i be a join node, j_1, j_2 be the left and right children of the node i respectively. $X_i = X_{j_1} = X_{j_2}$ and there are no edges between $V(T_{j_1}) \setminus X_i$ and $V(T_{j_2}) \setminus X_i$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple at node i . If Ψ is invalid we set $M_i[\Psi] = 0$. Otherwise, we compute $M_i[\Psi]$ value as follows:

Definition 25. Let $s = (w_1, w_2, w_3, \dots, w_r)$, $s' = (w'_1, w'_2, w'_3, \dots, w'_r)$ and $s'' = (w''_1, w''_2, w''_3, \dots, w''_r)$ be three sequences. We say that $s = \text{Merge}(s', s'')$ if the following conditions are satisfied.

1. $\forall \ell w_\ell \in X_i \implies w'_\ell = w''_\ell = w_\ell$.
2. $\forall \ell w_\ell = \text{FG} \implies \text{either } (w'_\ell = \text{FG} \text{ and } w''_\ell = \text{DC}) \text{ or } (w'_\ell = \text{DC} \text{ and } w''_\ell = \text{FG})$.
3. $\forall \ell w_\ell = \text{DC} \implies w'_\ell = w''_\ell = \text{DC}$.

Note that, if $s' \in \Gamma_{A_{j_1}}$ and $s'' \in \Gamma_{A_{j_2}}$ are legal sequences at node j_1 and j_2 respectively then s is a legal sequence at node i with respect to A . We extend the Merge operation to sets of sequences as follows:

$$\text{Merge}(S_1, S_2) = \{s \mid \exists s' \in S_1, s'' \in S_2 \text{ such that } s = \text{Merge}(s', s'')\}.$$

We set $M_i[\Psi] = 1$ if there exists $P_{j_1}, Q_{j_1}, P_{j_2}$ and Q_{j_2} such that the following conditions are satisfied:

- (i) $P_i = \text{Merge}(P_{j_1}, P_{j_2})$,
- (ii) $Q_i = \text{Merge}(Q_{j_1}, Q_{j_2})$,
- (iii) $M_{j_1}[(A_i, B_i, P_{j_1}, Q_{j_1})] = 1$, and
- (iv) $M_{j_2}[(A_i, B_i, P_{j_2}, Q_{j_2})] = 1$.

The graph has valid bipartitioning if there exists a Ψ such that $M_r[\Psi] = 1$. Where r is the root node of the nice tree decomposition. The correctness of the algorithm is implied by the correctness of $M_i[\Psi]$ values, which can be proved using a bottom up induction on the nice tree decomposition. The time complexity at each of the nodes in the tree decomposition is as follows: constant time at leaf nodes, $O(2^{O(t^r)})$ time at insert nodes, $O(2^{O(t^r)})$ time at forget nodes and $O(2^{O(t^r)})$ time at join nodes. Thus we get the following:

Theorem 11. *There is an $O(2^{O(t^r)} \cdot n)$ time algorithm that solves the H-FREE 2-COLORING problem for any arbitrary fixed H ($|V(H)| = r$), on graphs with tree-width at most t .*

4.2.3 H -FREE q -COLORING

We note that techniques used in 4.2.2 extend in a straightforward manner to solve the H -FREE q -COLORING problem. We discuss the algorithm for completeness. Here we consider tuples Ψ that have $2q$ sets. That is $\Psi = (A_i^1, A_i^2, \dots, A_i^q, P_i^1, P_i^2, \dots, P_i^q)$.

We perform dynamic programming over the nice tree decomposition. At each node i we guess a partition $(A_i^1, A_i^2, \dots, A_i^q)$ of X_i and possible induced subgraphs of H that are part of A_i^z for $1 \leq z \leq q$ respectively. We check if such a partition is possible. Below we explain the algorithm in detail.

Let the vertices of the graph H are labeled as $u_1, u_2, u_3, \dots, u_r$. Let $(A_i^1, A_i^2, \dots, A_i^q)$ be a partition of vertices in the bag X_i . Let (A^1, A^2, \dots, A^q) be a partition of $V(T_i)$ such that for $1 \leq z \leq q$, $A^z \supseteq A_i^z$. For $1 \leq z \leq q$, we define $\Gamma_{A_i^z}$ as follows:

$$\begin{aligned} S_{A_i^z} &= \{(w_1, w_2, w_3, \dots, w_r) \mid w_\ell \in \{A_i^z \cup \{\text{FG}, \text{DC}\}\}, \\ &\quad \forall \ell_1 \neq \ell_2, w_{\ell_1} = w_{\ell_2} \implies w_{\ell_1} \in \{\text{FG}, \text{DC}\}\}, \\ I_{A_i^z} &= \{s = (w_1, w_2, w_3, \dots, w_r) \in S_{A_i^z} \mid \text{there exists } \ell_1 \neq \ell_2 \\ &\quad \text{such that } w_{\ell_1} = \text{FG}, w_{\ell_2} = \text{DC} \text{ and } \{u_{\ell_1}, u_{\ell_2}\} \in E(H)\} \\ \Gamma_{A_i^z} &= S_{A_i^z} \setminus I_{A_i^z} \end{aligned}$$

Here FG represents a vertex in $A^z \setminus A_i^z$, the forgotten vertices in A^z and DC stands for don't care. That is we don't care if the corresponding vertex is part of the subgraph or not.

A sequence in $S_{A_i^z}$ corresponds to a subgraph H' of H in A^z as follows:

1. If $w_\ell = \text{FG}$ then u_ℓ is part of $A^z \setminus A_i^z$, the forgotten vertices in A^z .
2. If $w_\ell = \text{DC}$ then u_ℓ need not be part of the subgraph H' .
3. If $w_\ell \in A_i^z$ then the vertex w_ℓ corresponds to the vertex u_ℓ of H' .

$\Gamma_{A_i^z}$ is the set of sequences that can become H in future at some ancestral (insert/join) node of the tree decomposition. Note that the sequences $I_{A_i^z}$ are excluded from $\Gamma_{A_i^z}$ because a forgot vertex cannot have an edge to a vertex which will come in future at some ancestral node (insert or join nodes).

Definition 26 (Induced Subgraph Legal Sequence in $\Gamma_{A_i^z}$ with respect to A^z for $1 \leq z \leq q$). *A sequence $s = (w_1, w_2, w_3, \dots, w_r) \in \Gamma_{A_i^z}$ is legal if the sequence s corresponds to subgraph H' of H within A^z as follows.*

Let $FV(s) = \{\ell | w_\ell = FG\}$, $DC(s) = \{\ell | w_\ell = DC\}$ and $VI(s) = [r] \setminus \{FV(s) \cup DC(s)\}$. Let H' be the induced subgraph of H formed by u_ℓ , $\ell \in \{VI(s) \cup FV(s)\}$. That is $H' = H[\{u_\ell | \ell \in VI(s) \cup FV(s)\}]$.

If there exist $|FV(s)|$ distinct vertices $z_\ell \in A^z \setminus A_i^z$ corresponding to each index in $FV(s)$ such that H' is isomorphic to $G[\{w_\ell | \ell \in VI(s)\} \cup \{z_\ell | \ell \in FV(s)\}]$, then s is legal. Otherwise, the sequence is illegal.

Let $\Psi = (A_i^1, A_i^2, \dots, A_i^q, P_i^1, P_i^2, \dots, P_i^q)$ be a tuple. Here, $(A_i^1, A_i^2, \dots, A_i^q)$ is a partition of X_i , $P_i^z \subseteq \Gamma_{A_i^z}$ for $1 \leq z \leq q$.

We define $M_i[\Psi]$ to be 1 if there is a partition (A^1, A^2, \dots, A^q) of $V(T_i)$ such that:

1. $A_i^z \subseteq A^z$ for $1 \leq z \leq q$.
2. Every sequence in P_i^z is legal with respect to A^z for $1 \leq z \leq q$.
3. Every sequence in $\Gamma_{A_i^z} \setminus P_i^z$ is illegal with respect to A^z for $1 \leq z \leq q$.
4. For $1 \leq z \leq q$, every A_i^z is H -free.

Otherwise $M_i[\Psi]$ is set to 0.

We call a tuple Ψ as invalid if one of the following condition occur. If Ψ is invalid we set $M_i[\Psi]$ to 0.

1. There exists a sequence $s \in P_i^z$ for some $1 \leq z \leq q$, such that s does not contain DC.

Now we explain how to compute $M_i[\Psi]$ values at the leaf, introduce, forgot and join nodes of the nice tree decomposition.

Leaf node: Let i be a leaf node, $X_i = \emptyset$, for $\Psi = (A_i^1, A_i^2, \dots, A_i^q, P_i^1, P_i^2, \dots, P_i^q)$, we have $M_i[\Psi] = 1$. Here $A_i^z = \emptyset$, $P_i^z \subseteq \{([DC]^r)\}$.

Introduce node: Let i be an introduce node and j be the child node of i . Let $\{v\} = X_i \setminus X_j$. Let $\Psi = (A_i^1, A_i^2, \dots, A_i^q, P_i^1, P_i^2, \dots, P_i^q)$ be a tuple at node i . If Ψ is invalid we set $M_i[\Psi] = 0$. Otherwise without loss of generality let us assume $v \in A_i^z$ for some $1 \leq z \leq q$.

$v \in A_i^z$: We set $M_i[\Psi] = 0$, if there exists an illegal sequence s (in P_i^z) containing v or if there exists a trivial legal sequence s containing v but s is not in P_i^z .

That is, we set $M_i[\Psi] = 0$ in one of the following (#) conditions occurs:

[(#) Conditions]

1. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} \in A_i^z$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H)$ but $\{v, w_{\ell_2}\} \notin E(G)$.
2. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} \in A_i^z$, $\{u_{\ell_1}, u_{\ell_2}\} \notin E(H)$ but $\{v, w_{\ell_2}\} \in E(G)$.
3. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} = \text{FG}$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H)$.
4. Let $s = (w_1, w_2, w_3, \dots, w_r) \in \Gamma_{A_i^z} \setminus P_i^z$. There exists ℓ_1 such that $w_{\ell_1} = v$ and for all $\ell_2 \neq \ell_1$ $w_{\ell_2} \in A_i^z \cup \{\text{DC}\}$. For all $\ell_1 \neq \ell_2$ $w_{\ell_1}, w_{\ell_2} \in A_i^z$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H) \iff \{w_{\ell_1}, w_{\ell_2}\} \in E(G)$.

Otherwise we set $M_i[\Psi] = M_j[\Psi']$, where Ψ' is obtained by replacing A_i^z with $A_i^z \setminus \{v\}$ and P_i^z with P_j^z which is computed as follows:

Definition 27. $\text{Rep}_{\text{DC}}(s, v) = s'$, sequence s' obtained by replacing v (if present) with DC in s .

Note that, $\text{Rep}_{\text{DC}}(s, v) = s$, if v not present in s .

$$P_j^z = \cup_{s \in P_i^z} \{\text{Rep}_{\text{DC}}(s, v)\}.$$

Forget node: Let i be a forget node and j be the only child of node i . Let $\{v\} = X_j \setminus X_i$. Let $\Psi = (A_i^1, A_i^2, \dots, A_i^q, P_i^1, P_i^2, \dots, P_i^q)$ be a tuple at node i . If Ψ is invalid we set $M_i[\Psi] = 0$. Otherwise, we set $M_i[\Psi] = \max_{1 \leq z \leq q} \{\delta_z\}$ where δ_z is computed as follows:

Computing δ_z : Set $A_j^z = A_i^z \cup \{v\}$. As v is the extra vertex in A_j^z , there could be many possible P_j^z at node j .

Definition 28. $\text{Rep}_{\text{FG}}(s, v) = s'$, sequence s' obtained by replacing v (if present) with FG in s .

Note that, if s does not contain the vertex v then $\text{Rep}_{\text{FG}}(s, v) = s$.

We also extend the definition of Rep_{FG} to a set of sequences as follows:

$$\text{Rep}_{\text{FG}}(S, v) = \cup_{s \in S} \{\text{Rep}_{\text{FG}}(s, v)\}.$$

Note that, if s is a legal sequence at the node j with respect to A_j^z , then $\text{Rep}_{\text{FG}}(s, v)$ is also a legal sequence at node i with respect to A_i^z .

$$\delta_z = \max_{\substack{P_j^z \subseteq \Gamma_{A_j^z} \\ \text{Rep}_{\text{FG}}(P_j^z, v) = P_i^z}} \{M_j[(A_j^1, A_j^2, \dots, A_j^q, P_j^1, P_j^2, \dots, P_j^q)]\}$$

Join node: Let i be a join node, j_1, j_2 be the left and right children of the node i respectively. $X_i = X_{j_1} = X_{j_2}$ and there are no edges between $V(T_{j_1}) \setminus X_i$ and $V(T_{j_2}) \setminus X_i$. Let $\Psi = (A_i^1, A_i^2, \dots, A_i^q, P_i^1, P_i^2, \dots, P_i^q)$ be a tuple at node i . If Ψ is invalid we set $M_i[\Psi] = 0$. Otherwise, we compute $M_i[\Psi]$ value as follows:

Definition 29. Let $s = (w_1, w_2, w_3, \dots, w_r)$, $s' = (w'_1, w'_2, w'_3, \dots, w'_r)$ and $s'' = (w''_1, w''_2, w''_3, \dots, w''_r)$ be three sequences. We say that $s = \text{Merge}(s', s'')$ if the following conditions are satisfied.

1. $\forall \ell w_\ell \in X_i \implies w'_\ell = w''_\ell = w_\ell$.
2. $\forall \ell w_\ell = \text{FG} \implies \text{either } (w'_\ell = \text{FG} \text{ and } w''_\ell = \text{DC}) \text{ or } (w'_\ell = \text{DC} \text{ and } w''_\ell = \text{FG})$.
3. $\forall \ell w_\ell = \text{DC} \implies w'_\ell = w''_\ell = \text{DC}$.

Note that, if $s' \in \Gamma_{A_{j_1}^z}$ and $s'' \in \Gamma_{A_{j_2}^z}$ are legal sequences at node j_1 and j_2 respectively then s is a legal sequence at node i with respect to A_i^z . We extend the Merge operation to sets of sequences as follows:

$$\text{Merge}(S_1, S_2) = \{s \mid \exists s' \in S_1, s'' \in S_2 \text{ such that } s = \text{Merge}(s', s'')\}.$$

We set $M_i[\Psi] = 1$ if there exists $P_{j_1}^z$ and $P_{j_2}^z$ for $1 \leq z \leq q$ such that the following conditions are satisfied:

- $P_i^z = \text{Merge}(P_{j_1}^z, P_{j_2}^z)$ for $1 \leq z \leq q$,
- $M_{j_1}[(A_{j_1}^1, A_{j_1}^2, \dots, A_{j_1}^q, P_{j_1}^1, P_{j_1}^2, \dots, P_{j_1}^q)] = 1$, and
- $M_{j_2}[(A_{j_2}^1, A_{j_2}^2, \dots, A_{j_2}^q, P_{j_2}^1, P_{j_2}^2, \dots, P_{j_2}^q)] = 1$.

The graph has valid bipartitioning if there exists a Ψ such that $M_r[\Psi] = 1$. Where r is the root node of the nice tree decomposition. The correctness of the algorithm is implied by the correctness of $M_i[\Psi]$ values, which can be proved using a bottom up induction on the nice tree decomposition. The time complexity of the algorithm is $O^*(q^{O(t^r)})$. Thus we state the following theorem.

Theorem 12. *There is an $O(q^{O(t^r)} \cdot n)$ time algorithm that solves the H-FREE q -COLORING problem for any arbitrary fixed H ($|V(H)| = r$), on graphs with tree-width at most t .*

For graphs with tree-width t , $\chi(H, G) \leq \chi(G) \leq t + 1$. Our techniques can also be used to compute the H -FREE CHROMATIC NUMBER of the graph by searching for the smallest q for which there is an H -free q -coloring. We have the following theorem.

Theorem 13. *There is an $O(t^{O(t^r)} \cdot n \log t)$ time algorithm to compute H -FREE CHROMATIC NUMBER of the graph whose tree-width is at most t .*

This shows that H -FREE CHROMATIC NUMBER problem is in FPT with respect to the parameter tree-width.

4.3 H -(Subgraph)Free Coloring

Before discussing the algorithm for the general H -(SUBGRAPH)FREE q -COLORING problem, we discuss algorithms for C_4 -(SUBGRAPH)FREE 2-COLORING, $\{K_r \setminus e\}$ -(SUBGRAPH)FREE 2-COLORING and C_r -(SUBGRAPH)FREE 2-COLORING and H -(SUBGRAPH)FREE 2-COLORING problems. Finally we discuss the algorithm for the general H -(SUBGRAPH)FREE q -COLORING problem.

4.3.1 C_4 -(SUBGRAPH)FREE 2-COLORING

A cycle of length 4 is formed when a pair of (adjacent or non-adjacent) vertices have two or more common neighbors. If a graph has no C_4 then any vertex pair can have at most one common neighbor. Let X_i be a bag at the node i of the nice tree decomposition. We guess a partition (A_i, B_i) of the bag X_i . For each pair of vertices from A_i (similarly B_i), we also guess if the pair has exactly one common forgotten neighbor in part A (similarly B) of the partition. We check if the above guesses lead to a valid partitioning in the subgraph $G[T_i]$, which is the graph induced by the vertices in the node i and all its descendent nodes. Below we formally explain the technique.

Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple defined as follows: (A_i, B_i) is a partition of X_i , $P_i \subseteq \binom{A_i}{2}$ and $Q_i \subseteq \binom{B_i}{2}$. Intuitively, P_i and Q_i are the set of those pairs that have exactly one common forgotten neighbor.

We define $M_i[\Psi]$ to be 1 if there is a partition (A, B) of $V(T_i)$ such that:

1. $A_i \subseteq A$ and $B_i \subseteq B$.
2. Every pair in P_i has exactly one common neighbor in $A \setminus A_i$.
3. Every pair in $\binom{A_i}{2} \setminus P_i$ does not have a common neighbor in $A \setminus A_i$.
4. Every pair in Q_i has exactly one common neighbor in $B \setminus B_i$.

5. Every pair in $\binom{B_i}{2} \setminus Q_i$ does not have a common neighbor in $B \setminus B_i$.

6. $G[A]$ and $G[B]$ do not have C_4 as a subgraph.

Otherwise, $M_i[\Psi]$ is set to 0. Suppose there exists a 4-tuple Ψ such that $M_r[\Psi] = 1$, where r is the root of the nice tree decomposition. Then the above conditions 1 and 6 ensure that G can be partitioned in the required manner.

When one of the following occurs, it is easy to see that the 4-tuple does not lead to a required partition. We say that the 4-tuple Ψ is *invalid* if one of the below cases occur:

- (i) $G[A_i]$ or $G[B_i]$ contains a C_4 .
- (ii) There exists a pair $\{x, y\} \in P_i$ with a common neighbor in A_i .
- (iii) There exists a pair $\{x, y\} \in Q_i$ with a common neighbor in B_i .

Note that it is easy to check if a given Ψ is invalid. Below we explain how to compute $M_i[\Psi]$ value at each node i .

Leaf node: For a leaf node i , $\Psi = (\emptyset, \emptyset, \emptyset, \emptyset)$ and $M_i[\Psi] = 1$.

Introduce node: Let j be the only child of the node i . Suppose $v \in X_i$ is the new vertex present in X_i , $v \notin X_j$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple of X_i . If Ψ is invalid, we set $M_i[\Psi]$ to 0. Otherwise, we use the following cases to compute the $M_i[\Psi]$ value.

Case 1, $v \in A_i$: If $\exists\{v, x\} \in P_i$ for some $x \in A_i$ or if $\exists\{x, y\} \in P_i$ such that $\{x, y\} \subseteq N(v) \cap A_i$, then $M_i[\Psi] = 0$. Otherwise, $M_i[\Psi] = M_j[\Psi']$, where $\Psi' = (A_i \setminus v, B_i, P_i, Q_i)$.

As v is a newly introduced vertex, it cannot have any forgotten neighbors. Hence, $\{v, x\} \in P_i \implies M_i[\Psi] = 0$. If x and y have a common forgotten neighbor, they all form a C_4 , together with v . Hence $\{x, y\} \in P_i \implies M_i[\Psi] = 0$.

Case 2, $v \in B_i$: If $\exists\{v, x\} \in Q_i$ for some $x \in B_i$ or if $\exists\{x, y\} \in Q_i$ such that $\{x, y\} \subseteq N(v) \cap B_i$, then $M_i[\Psi] = 0$. Otherwise, $M_i[\Psi] = M_j[\Psi']$, where $\Psi' = (A_i, B_i \setminus v, P_i, Q_i)$.

Forget node: Let j be the only child of the node i . Suppose $v \in X_j$ is the vertex missing in X_i , $v \notin X_i$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple of X_i . If Ψ is invalid, we set $M_i[\Psi]$ to 0. Otherwise, $M_i[\Psi]$ is computed as follows:

Case 1, $v \in A_j$: If $\exists x, y \in A_i$ such that $xv, yv \in E(G)$, then v is a common forgotten neighbor for x and y . Hence we set $M_i[\Psi] = 0$ whenever $\{x, y\} \notin P_i$. Otherwise, let $R = \{\{x, y\} | x, y \in A_i \cap N(v)\}$. At node j , note that any pair in R with a common forgotten neighbor will form a C_4 . Hence we consider only those P_j 's that

are disjoint with R . Also there can be new pairs formed with v at the node j . Let $S = \{\{v, x\} | x \in A_i\}$. We have the following equation.

$$\delta_1 = \max_{X \subseteq S} \{M_j[(A_i \cup v, B_i, (P_i \setminus R) \cup X, Q_i)]\}.$$

Case 2, $v \in B_j$: This is analogous to Case 1. We set $M_i[\Psi] = 0$, whenever $\{x, y\} \notin Q_i$. Otherwise, let $R = \{\{x, y\} | x, y \in B_i \cap N(v)\}$ and $S = \{\{v, x\} | x \in B_i\}$.

$$\delta_2 = \max_{X \subseteq S} \{M_j[(A_i, B_i \cup v, P_i, (Q_i \setminus R) \cup X)]\}.$$

If $M_i[\Psi]$ is not set to 0 already, we set $M_i[\Psi] = \max\{\delta_1, \delta_2\}$.

Join node: Let j_1 and j_2 be the children of the node i . By the property of nice tree decomposition, we have $X_i = X_{j_1} = X_{j_2}$ and $V(T_{j_1}) \cap V(T_{j_2}) = X_i$. There are no edges between $V(T_{j_1}) \setminus X_i$ and $V(T_{j_2}) \setminus X_i$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple of X_i . If Ψ is invalid, we set $M_i[\Psi]$ to 0. Otherwise, we use the following expression to compute the value of $M_i[\Psi]$.

A pair $\{x, y\} \in P_i$ can come either from the left subtree or from the right subtree but not from both, for that would imply two distinct common neighbors for x and y and hence a C_4 . For $X \subseteq P_i$ and $Y \subseteq Q_i$, $\Psi_1 = (A_i, B_i, X, Y)$ and $\Psi_2 = (A_i, B_i, P_i \setminus X, Q_i \setminus Y)$.

$$M_i[\Psi] = \begin{cases} 1, & \exists X \subseteq P_i, Y \subseteq Q_i \text{ such that } M_{j_1}[\Psi_1] = M_{j_2}[\Psi_2] = 1. \\ 0, & \text{Otherwise.} \end{cases}$$

The correctness of the algorithm is implied by the correctness of $M_i[\Psi]$ values, which follows by a bottom-up induction on the nice tree decomposition. G has a valid bipartitioning if there exists a 4-tuple Ψ such that $M_r[\Psi] = 1$, where r is the root of the nice tree decomposition.

The time complexity at each of the nodes in the tree decomposition is as follows: constant time at leaf nodes, $O(2^{t+t^2})$ time at insert nodes, $O(2^{2t+t^2})$ time at forget nodes and $O(2^{t+2t^2})$ time at join nodes. This gives the following:

Theorem 14. *There is an $O(2^{O(t^2)}n)$ time algorithm that solves the C_4 -(SUBGRAPH)FREE 2-COLORING problem on graphs with tree-width at most t .*

4.3.2 $\{K_r \setminus e\}$ -(SUBGRAPH)FREE 2-COLORING

Let X_i be a bag at the node i of the nice tree decomposition. Let $\Psi = (A_i, B_i, P_i, Q_i)$ is a 4-tuple defined as follows: (A_i, B_i) be a partition of X_i , $P_i \subseteq \binom{A_i}{r-2}$ and $Q_i \subseteq \binom{B_i}{r-2}$.

We define $M_i[\Psi]$ to be 1 if there is a partition (A, B) of $V(T_i)$ such that:

1. $A_i \subseteq A$ and $B_i \subseteq B$.
2. For every set S in P_i , there is exactly one vertex $v \in (V(T_i) \setminus X_i) \cap A$, such that $G[S \cup v]$ is a K_{r-1} .
3. For every set $S \in \binom{A_i}{r-2} \setminus P_i$, $G[S \cup v]$ is not K_{r-1} for any choice of vertex $v \in (V(T_i) \setminus X_i) \cap A$.
4. For every set S in Q_i , there is exactly one vertex $v \in (V(T_i) \setminus X_i) \cap B$, such that $G[S \cup v]$ is a K_{r-1} .
5. For every set $S \in \binom{B_i}{r-2} \setminus Q_i$, $G[S \cup v]$ is not K_{r-1} for any choice of vertex $v \in (V(T_i) \setminus X_i) \cap B$.
6. $G[A]$ and $G[B]$ do not have $K_r \setminus e$ as a subgraph.

Otherwise, $M_i[\Psi]$ is set to 0.

We say that a 4-tuple is *invalid* if one of the following occurs:

- (i) $G[A_i]$ or $G[B_i]$ has $K_r \setminus e$ as subgraph.
- (ii) There exists a set Y in P_i such that every vertex in Y has a common neighbor in A_i .
- (iii) There exists a set Y in Q_i such that every vertex in Y has a common neighbor in B_i .
- (iv) There exists a set Y in P_i such that Y is not a clique.
- (v) There exists a set Y in Q_i such that Y is not a clique.

We calculate $M_i[\Psi]$ based on the type of node i .

Leaf node: For a leaf node, $\Psi = (\emptyset, \emptyset, \emptyset, \emptyset)$ and $M_i[\Psi] = 1$.

Introduce node: Let j be the only child of node i . Suppose v is the lone vertex in $X_i \setminus X_j$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple of X_i . If Ψ is invalid, we set $M_i[\Psi]$ to 0. Otherwise, we use the following cases to compute $M_i[\Psi]$ value.

Case 1, $v \in A_i$: If $\exists Y \in P_i$ such that $Y \subseteq (N[v] \cap A_i)$, $M_i[\Psi] = 0$. Otherwise, $M_i[\Psi] = M_j[\Psi']$, where $\Psi' = (A_i \setminus v, B_i, P_i, Q_i)$.

Case 2, $v \in B_i$: If $\exists Y \in Q_i$ such that $Y \subseteq (N[v] \cap B_i)$, $M_i[\Psi] = 0$. Otherwise, $M_i[\Psi] = M_j[\Psi']$, where $\Psi' = (A_i, B_i \setminus v, P_i, Q_i)$.

Forget node: Let j be the only child of the node i . Suppose v is the lone vertex in $X_j \setminus X_i$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple of X_i . If Ψ is invalid, we set $M_i[\Psi]$ to 0. Otherwise, $M_i[\Psi]$ is computed as follows.

Case 1: If $\exists Y \in \binom{A_i}{r-2}$ such that $Y \subseteq N(v)$ and $Y \notin P_i$, then $M_i[\Psi] = 0$. Otherwise, let $R = \{Y \in \binom{A_i}{r-2} | Y \subseteq N(v)\}$ and $S = \{Y \in \binom{A_i}{r-2} | v \in Y\}$.

$$\delta_1 = \max_{Z \subseteq S} \{M_j[(A_i \cup v, B_i, (P_i \setminus R) \cup Z, Q_i)]\}.$$

Case 2: If $\exists Y \in \binom{B_i}{r-2}$ such that $Y \subseteq N(v)$ and $Y \notin Q_i$, then $M_i[\Psi] = 0$. Otherwise, let $R = \{Y \in \binom{B_i}{r-2} | Y \subseteq N(v)\}$ and $S = \{Y \in \binom{B_i}{r-2} | v \in Y\}$.

$$\delta_2 = \max_{Z \subseteq S} \{M_j[(A_i, B_i \cup v, P_i, (Q_i \setminus R) \cup Z)]\}.$$

If $M_i[\Psi]$ is not set to 0 already, we set $M_i[\Psi] = \max\{\delta_1, \delta_2\}$.

Join node: Let j_1 and j_2 be the children of node i . $X_i = X_{j_1} = X_{j_2}$ and $V(T_{j_1}) \cap V(T_{j_2}) = X_i$. There are no edges between $V(T_{j_1}) \setminus X_i$ and $V(T_{j_2}) \setminus X_i$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple of X_i . If Ψ is invalid, we set $M_i[\Psi]$ to 0. Otherwise, we use the following expression to compute $M_i[\Psi]$ value. For $Z_1 \subseteq P_i$ and $Z_2 \subseteq Q_i$, let $\Psi_1 = (A_i, B_i, Z_1, Z_2)$ and $\Psi_2 = (A_i, B_i, P_i \setminus Z_1, Q_i \setminus Z_2)$.

$$M_i[\Psi] = \begin{cases} 1, & \text{If } \exists Z_1 \subseteq P_i, Z_2 \subseteq Q_i \text{ such that } M_{j_1}[\Psi_1] = 1 \text{ and } M_{j_2}[\Psi_2] = 1. \\ 0, & \text{Otherwise.} \end{cases}$$

The correctness of the algorithm is implied by the correctness of $M_i[\Psi]$ values, which follows by a bottom-up induction on the nice tree decomposition. G has a valid bipartitioning if there exists a 4-tuple Ψ such that $M_r[\Psi] = 1$, where r is the root of the nice tree decomposition. The time complexity at each of the nodes in the tree decomposition is as follows: constant time at leaf nodes, $O(2^{t+t^{r-2}})$ time at insert nodes, $O(2^{t+t^{r-2}})$ time at forget nodes and $O(2^{t+2t^{r-2}})$ time at join nodes. With this we state the following theorem.

Theorem 15. *There is an $O^*(2^{O(t^{r-2})})$ time algorithm that solves the $\{K_r \setminus e\}$ -(SUBGRAPH)FREE 2-COLORING problem w , on graphs with tree-width at most t .*

We remark that the technique will also work for the case of $K_r \setminus \{e_1, e_2\}$ where e_1, e_2 two non-adjacent edges. The only difference in the algorithm is that the elements in the sets P_i and Q_i should not only include cliques of size $r - 2$ but also vertex sets that form $K_{r-2} \setminus e$.

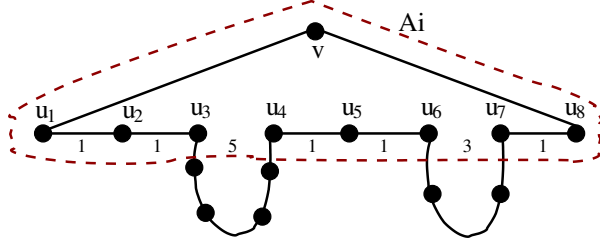


Figure 4.4: A cycle of length 15 formed with vertex v at an insert node. The vertices outside the dotted outline are forgotten vertices. The ℓ values are marked between the u_i 's in the figure.

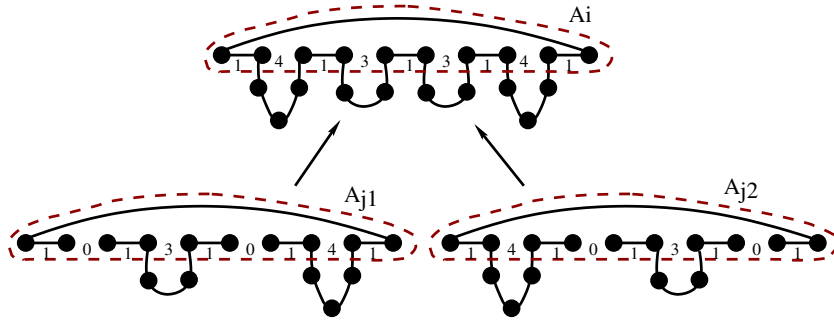


Figure 4.5: A cycle of length 20 formed using the paths from left and right subtrees at join node. The vertices outside the dotted outline are forgotten vertices.

4.3.3 C_r -(SUBGRAPH)FREE 2-COLORING

Let X_i be a bag at node i of the nice tree decomposition. Let (A_i, B_i) be a partition of X_i . We can easily check if $G[A_i]$ or $G[B_i]$ has a cycle of length r . Otherwise, we need to see if there is a partition (A, B) of $V(T_i)$ such that $A_i \subseteq A$, $B_i \subseteq B$ and both $G[A]$ and $G[B]$ do not have cycle of length r . If there is such a partition (A, B) , then $G[A_i]$ and $G[B_i]$ may have partial paths of length at most $r - 2$ which can lead to cycle of length r at some ancestral node j (insert node or join node) of the nice tree decomposition (see Figures 4.4 and 4.5). We perform dynamic programming over the nice tree decomposition. At each node i , we guess a partition (A_i, B_i) of X_i and possible partial paths of length at most $r - 2$ of A and B using vertices of A_i and B_i respectively. We check if such a partition (A, B) is possible or not. Below we explain the algorithm in detail.

Below, we define Γ_{A_i} and Γ_{B_i} which are the set of all possible partial paths that need to be considered within the parts A_i and B_i respectively. The members of Γ_{A_i} and Γ_{B_i} are called sequences.

$$\Gamma_{A_i} = \{(u_1, \ell_1, u_2, \ell_2, \dots, u_{r-3}, \ell_{r-3}, u_{r-2}) \mid u_1, u_2, \dots, u_{r-2} \in A_i, \\ 2 \leq \sum_{j=1}^{r-2} \ell_j \leq r - 2, 0 \leq \ell_1, \dots, \ell_{r-2} \leq r - 2, \max\{\ell_1, \dots, \ell_{r-2}\} \geq 2\}.$$

$$\Gamma_{B_i} = \{(u_1, \ell_1, u_2, \ell_2, \dots, u_{r-3}, \ell_{r-3}, u_{r-2}) \mid u_1, u_2, \dots, u_{r-2} \in B_i, \\ 2 \leq \sum_{j=1}^{j=r-2} \ell_j \leq r-2, 0 \leq \ell_1, \dots, \ell_{r-2} \leq r-2, \max\{\ell_1, \dots, \ell_{r-2}\} \geq 2\}.$$

Definition 30 (Legal Sequence in Γ_{A_i} with respect to A). A sequence $s = (u_1, \ell_1, u_2, \ell_2, \dots, u_{r-3}, \ell_{r-3}, u_{r-2})$ in Γ_{A_i} is said to be legal with respect to a set $A \supseteq A_i$, if the following conditions are true for each $1 \leq i \leq r-3$:

- (a) If $\ell_i > 1$, there is a path of length ℓ_i from u_i to u_{i+1} . Except for u_i and u_{i+1} , all the other $\ell_i - 1$ vertices in the path are vertices in $A \setminus A_i$. All the above paths from the sequence must be vertex disjoint.
- (b) If $\ell_i = 1$, then $u_i u_{i+1} \in E(G)$.
- (c) If $\ell_i = 0$, then either $u_i u_{i+1} \notin E(G)$ or the edge $u_i u_{i+1}$ is not included in the path.

Legal sequences in Γ_{B_i} with respect to B are analogously defined.

Intuitively, condition (a) above insists that all the intermediate vertices in the path from u_i to u_{i+1} are forgotten vertices at the node i .

Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple defined as follows: (A_i, B_i) is a partition of X_i , $P_i \subseteq \Gamma_{A_i}$ and $Q_i \subseteq \Gamma_{B_i}$. Let A, B be a partition of $V(T_i)$ such that $A_i \subseteq A$ and $B_i \subseteq B$. We define $M_i[\Psi]$ to be 1 if there is a partition (A, B) of $V(T_i)$ such that:

1. $A_i \subseteq A$ and $B_i \subseteq B$.
2. Every sequence $s \in P_i$ is legal w.r.t. A .
3. Every sequence $s \in Q_i$ is legal w.r.t. B .
4. Every sequence $s \in \Gamma_{A_i} \setminus P_i$ is illegal w.r.t. A .
5. Every sequence $s \in \Gamma_{B_i} \setminus Q_i$ is illegal w.r.t. B .
6. $G[A]$ and $G[B]$ do not have C_r as a subgraph.

Otherwise, $M_i[\Psi]$ is set to 0.

Like in the case of C_4 , when one of the following occurs, it is easy to see that the 4-tuple does not lead to a required partition. We say that the 4-tuple Ψ is *invalid* in these cases:

- (i) $G[A_i]$ or $G[B_i]$ contains a C_r .
- (ii) There exist an $s = (u_1, \ell_1, u_2, \ell_2, \dots, u_{r-3}, \ell_{r-3}, u_{r-2}) \in P_i$, $u \in A_i$ such that s and u form a C_r within A .

(iii) There exist an $s = (u_1, \ell_1, u_2, \ell_2, \dots, u_{r-3}, \ell_{r-3}, u_{r-2}) \in Q_i$, $u \in B_i$ such that s and u form a C_r within B .

Condition (i) can be verified in $O^*(t^r)$ time. For condition (ii) (and similarly for (iii)) we do the following. For a vertex $u \in A_i$ and a sequence s we can verify if u forms a C_r with s as follows. Consider all the edges adjacent to the vertices of the sequence s . For each pair of such edges, suppose the edges are adjacent to the vertices u_x and u_y (for $x < y$) we check if the subsequence $(u, 1, u_x, \ell_x, u_{x+1}, \dots, u_y, 1, u)$ forms a cycle of length r . For a vertex u and a sequence s we can verify this in $O(r^3)$ time. The total time complexity to check both conditions (ii) and (iii) is $O((|P_i| + |Q_i|)tr^3)$ time.

We calculate $M_i[\Psi]$ based on the type of node i . For algorithmic convenience, we add $2r - 4$ isolated vertices to the graph G . These vertices are also added to each bag of the nice tree decomposition. This will increase the tree-width by $2r - 4$. This changes some aspects of the decomposition but still enough “niceness” of the nice tree decomposition is retained for the purpose of the algorithm. Among the added vertices, a fixed set of $r - 2$ vertices added to every A_i and the other $r - 2$ vertices to B_i .

Leaf node: For a leaf node, note that there is only one partition (A_i, B_i) possible – where we add the fixed set of the $(r - 2)$ added vertices to each of A_i and B_i . For this partition, $\Gamma_{A_i} = \Gamma_{B_i} = \emptyset$, $\Psi = (A_i, B_i, \emptyset, \emptyset)$ and $M_i[\Psi] = 1$.

Introduce node: Let j be the only child of node i . Suppose $v \in X_i$ is the new vertex present in X_i , $v \notin X_j$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple of X_i . If Ψ is invalid, we set $M_i[\Psi]$ to 0. Otherwise we compute the $M_i[\Psi]$ value in the following manner. We only describe the case when $v \in A_i$.

Case $v \in A_i$: Suppose there is a sequence $s = (\dots, u_x, \ell_x, v, \ell_{x+1}, u_{x+2}, \dots) \in P_i$ containing v . Since v is newly introduced, it cannot have forgotten neighbors. Hence the following cases are illegal, and we can set the corresponding $M_i[\Psi]$ to 0 if any of these occur.

- $\ell_x \geq 2$.
- $\ell_{x+1} \geq 2$.
- $\ell_x = 1$ and $u_x v \notin E(G)$.
- $\ell_{x+1} = 1$ and $vu_{x+2} \notin E(G)$.

We will compute $M_i[\Psi]$ through $M_j[\Psi']$ where $\Psi' = (A_i \setminus v, B_i, P_j, Q_i)$. We compute P_j incrementally as follows: Initially $P_j = \emptyset$. For each sequence $s \in P_i$, we add the sequence(s) to P_j as follows:

- If v is not part of s , then add s to P_j .
- If $s = (v, \ell_1, u_2, \dots)$, then add the sequence $(z, 0, u_2, \dots)$ to P_j , where z is a vertex not in s but in A_i . Such a vertex is always available as there are $r - 2$ isolated vertices in A_i .
- $s = (\dots, u_x, \ell_x, v, \ell_{x+1}, u_{x+2}, \dots)$, then add the sequence $(\dots, u_x, 0, z, 0, u_{x+2}, \dots)$ to P_j . As above z is an isolated vertex in A_i , but not in s .
- $s = (\dots, u_{r-3}, \ell_{r-3}, v)$, then add the sequence $(\dots, u_{r-3}, 0, z)$ to P_j , where z is an isolated vertex.

We set $M_i[\Psi] = M_j[\Psi']$.

The case when $v \in B_i$ is similar to the above case, but we process analogously on the sets B_i and Q_i .

Forget node: Let j be the only child of the node i . Suppose v is the lone vertex in $X_j \setminus X_i$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple of X_i . If Ψ is invalid, we set $M_i[\Psi]$ to 0. Otherwise, $M_i[\Psi]$ is computed as follows:

We first consider the case when $v \in A_j$ at node j . This means that $A_j = A_i \cup v$. As v is an extra vertex present in j , there are many choices for P_j . We compute $\delta_1 = \max_{P_j} \{M_j[\Psi']\}$, where $\Psi' = (A_j, B_i, P_j, Q_i)$.

To understand the possible choices for P_j , we need the following definition.

Definition 31. Let $s' \in \Gamma_{A_j}$ be a sequence at the child node j , such that s' contains v . Let $s' = (u_1, \ell_1, u_2, \dots, u_x, \ell_x, v, \ell_{x+1}, u_{x+2}, \dots, u_{r-2})$. Then $\text{Diff}(s', v)$ is the following set of sequences at node i obtained by starting from $s'' = (u_1, \ell_1, u_2, \dots, u_x, (\ell_x + \ell_{x+1}), u_{x+2}, \dots, u_{r-2})$, and performing one of the below 3 operations once. Here z is any vertex in A_i that is not present in s' already.

- Prefix with $(z, 0)$,
- Suffix with $(0, z)$.
- Replace any 0 with $(0, z, 0)$.

We also define $\text{Diff}(S, v)$ for a set of sequences S as follows:

$$\text{Diff}(S, v) = \cup_{s' \in S} \{\text{Diff}(s', v)\}.$$

Note that if $s' \in \Gamma_{A_j}$ is a legal sequence w.r.t. some $A \supseteq A_j$, then every $s'' \in \text{Diff}(s', v)$ is also legal w.r.t. the same set A .

Let $S_v = \{s \in \Gamma_{A_j} \mid v \text{ appears in the sequence } s\}$. We compute δ_1 as follows.

$$\delta_1 = \max_{\substack{Z_1 \subseteq P_i, Z_2 \subseteq S_v \\ Z_1 \cup \text{Diff}(Z_2, v) = P_i}} \{M_j[(A_j, B_i, Z_1 \cup Z_2, Q_i)]\}.$$

We analogously compute δ_2 for the case when $v \in B_j$ (where $B_j = B_i \cup v$) at node j . If $M_i[\Psi]$ is not set to 0 already, we set $M_i[\Psi] = \max\{\delta_1, \delta_2\}$.

Join node: Let j_1 and j_2 be the children of the node i . $X_i = X_{j_1} = X_{j_2}$ and $V(T_{j_1}) \cap V(T_{j_2}) = X_i$. There are no edges between $V(T_{j_1}) \setminus X_i$ and $V(T_{j_2}) \setminus X_i$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple of X_i . If Ψ is invalid, we set $M_i[\Psi]$ to 0. Otherwise, we use the following expression to compute $M_i[\Psi]$ value.

Definition 32. Let $s = (u_1, \ell_1, u_2, \ell_2, \dots, u_{r-3}, \ell_{r-3}, u_{r-2})$, $s' = (u_1, \ell'_1, u_2, \ell'_2, \dots, u_{r-3}, \ell'_{r-3}, u_{r-2})$ and $s'' = (u_1, \ell''_1, u_2, \ell''_2, \dots, u_{r-3}, \ell''_{r-3}, u_{r-2})$ be three sequences. We say that $s = \text{Merge}(s', s'')$ if for all $1 \leq i \leq r-3$, the below conditions are satisfied:

- $\ell_i \in \{0, 1\} \iff \ell'_i = \ell''_i = \ell_i$.
- $\ell_i > 1 \iff$ either $(\ell'_i = \ell_i \text{ and } \ell''_i = 0)$ or $(\ell'_i = 0 \text{ and } \ell''_i = \ell_i)$.

Note that if $s' \in \Gamma_{A_{j_1}}$ and $s'' \in \Gamma_{A_{j_2}}$ are legal sequences w.r.t. some A , then the sequence $s = \text{Merge}(s_1, s_2)$ is a legal sequence w.r.t the same A . We extend the Merge operation to sets as follows:

$$\text{Merge}(P_{j_1}, P_{j_2}) = \{s \mid \exists s' \in P_{j_1}, s'' \in P_{j_2} \text{ such that } s = \text{Merge}(s', s'')\}.$$

Note that $P_{j_1}, P_{j_2} \subseteq \text{Merge}(P_{j_1}, P_{j_2})$ because given a sequence $s' \in P_{j_1}$, we can construct an $s'' \in P_{j_2}$ with the same vertices and ordering, but with all ℓ_i values set to 0. This will yield $s' = \text{Merge}(s', s'')$.

We set $M_i[\Psi] = 1$ if there exist $P_{j_1}, Q_{j_1}, P_{j_2}$ and Q_{j_2} such that all the following conditions are satisfied:

- (i) $P_i = \text{Merge}(P_{j_1}, P_{j_2})$,
- (ii) $Q_i = \text{Merge}(Q_{j_1}, Q_{j_2})$,
- (iii) $M_{j_1}[(A_i, B_i, P_{j_1}, Q_{j_1})] = 1$, and
- (iv) $M_{j_2}[(A_i, B_i, P_{j_2}, Q_{j_2})] = 1$.

Otherwise we set $M_i[\Psi] = 0$.

As in the case of $H = C_4$, the correctness of the algorithm is implied by the correctness of $M_i[\Psi]$ values, which follows by a bottom-up induction on the nice tree decomposition. G has a valid bipartitioning if there exists a 4-tuple Ψ such that $M_r[\Psi] = 1$, where r is the root of the nice tree decomposition.

Note that $|\Gamma_{A_i}| = |\Gamma_{B_i}| \leq (t+1)^{r-2}(r-2)!(r-1)^{r-3} = O((tr^2)^{r-2})$. The time complexity at each of the nodes in the tree decomposition is dominated by the time complexity at the join node, which is $O^*(2^{O((tr^2)^{r-2})})$. Thus we get the following:

Theorem 16. *There is an $2^{O((tr^2)^{r-2})} \cdot n^{O(1)}$ time algorithm that solves the C_r -(SUBGRAPH)FREE 2-COLORING problem, on graphs with tree-width at most t .*

4.3.4 H -(SUBGRAPH)FREE 2-COLORING

The techniques described in Section 4.2.2 can also be used to solve the H -(SUBGRAPH)FREE 2-COLORING. As we are looking for bipartitioning without H as a subgraph we need to modify the Definition 22 and (\star) conditions. Instead of Definition 22 we have Definition 33.

Definition 33 (Subgraph Legal Sequence in Γ_{A_i} with respect to A). *A sequence $s = (w_1, w_2, w_3, \dots, w_r) \in \Gamma_{A_i}$ is legal if the sequence s corresponds to subgraph H' of H within A as follows.*

Let $FV(s) = \{\ell | w_\ell = FG\}$, $DC(s) = \{\ell | w_\ell = DC\}$ and $VI(s) = [r] \setminus \{FV(s) \cup DC(s)\}$. Let H' be the induced subgraph of H formed by u_ℓ , $\ell \in \{VI(s) \cup FV(s)\}$. That is $H' = H[\{u_\ell | \ell \in VI(s) \cup FV(s)\}]$.

If there exist $|FV(s)|$ distinct vertices $z_\ell \in A \setminus A_i$ corresponding to each index in $FV(s)$ such that H' is subgraph of $G[\{w_\ell | \ell \in VI(s)\} \cup \{z_\ell | \ell \in FV(s)\}]$, then s is legal. Otherwise, the sequence is illegal.

At the introduced node, instead of (\star) conditions we have to check the following $(\star\star)$ conditions:

[$\star\star$ Conditions]

1. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} \in A_i$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H)$ but $\{v, w_{\ell_2}\} \notin E(G)$.
2. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} = FG$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H)$.
3. Let $s = (w_1, w_2, w_3, \dots, w_r) \in \Gamma_{A_i} \setminus P_i$. There exists ℓ_1 such that $w_{\ell_1} = v$ and for all $\ell_2 \neq \ell_1$ $w_{\ell_2} \in A_i \cup \{DC\}$. For all $\ell_1 \neq \ell_2$ $w_{\ell_1}, w_{\ell_2} \in A_i$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H) \implies \{w_{\ell_1}, w_{\ell_2}\} \in E(G)$.

Thus we get the following:

Theorem 17. *There is an $2^{O(t^r)} \cdot n$ time algorithm that solves the H -(SUBGRAPH)FREE 2-COLORING problem for any arbitrary fixed H ($|V(H)| = r$), on graphs with tree-width at most t .*

4.3.5 H -(SUBGRAPH)FREE q -COLORING

The techniques described in Section 4.2.3 can also be used to solve the H -(SUBGRAPH)FREE q -COLORING. We need to modify the Definition 26 and (#) conditions. Instead of Definition 26 we have Definition 34.

Definition 34 (Subgraph Legal Sequence in $\Gamma_{A_i^z}$ with respect to A^z for $1 \leq z \leq q$). *A sequence $s = (w_1, w_2, w_3, \dots, w_r) \in \Gamma_{A_i^z}$ is legal if the sequence s corresponds to subgraph H' of H within A^z as follows.*

Let $FV(s) = \{\ell | w_\ell = FG\}$, $DC(s) = \{\ell | w_\ell = DC\}$ and $VI(s) = [r] \setminus \{FV(s) \cup DC(s)\}$. Let H' be the induced subgraph of H formed by u_ℓ , $\ell \in \{VI(s) \cup FV(s)\}$. That is $H' = H[\{u_\ell | \ell \in VI(s) \cup FV(s)\}]$.

If there exist $|FV(s)|$ distinct vertices $z_\ell \in A^z \setminus A_i^z$ corresponding to each index in $FV(s)$ such that H' is subgraph of $G[\{w_\ell | \ell \in VI(s)\} \cup \{z_\ell | \ell \in FV(s)\}]$, then s is legal. Otherwise, the sequence is illegal.

At the introduced node, instead of (#) conditions we have to check the following (##) conditions:

[(##) Conditions]

1. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} \in A_i^z$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H)$ but $\{v, w_{\ell_2}\} \notin E(G)$.
2. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} = FG$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H)$.
3. Let $s = (w_1, w_2, w_3, \dots, w_r) \in \Gamma_{A_i^z} \setminus P_i^z$. There exists ℓ_1 such that $w_{\ell_1} = v$ and for all $\ell_2 \neq \ell_1$ $w_{\ell_2} \in A_i^z \cup \{DC\}$. For all $\ell_1 \neq \ell_2$ $w_{\ell_1}, w_{\ell_2} \in A_i^z$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H) \implies \{w_{\ell_1}, w_{\ell_2}\} \in E(G)$.

With this we state the following theorems:

Theorem 18. *There is an $O(q^{O(tr)} \cdot n)$ time algorithm that solves the H -(SUBGRAPH)FREE q -COLORING for any arbitrary fixed H ($|V(H)| = r$), on graphs with tree-width at most t .*

Theorem 19. *There is an $O(t^{O(tr)} \cdot n \log t)$ time algorithm to compute H -(SUBGRAPH)FREE CHROMATIC NUMBER of the graph whose tree-width is at most t .*

Chapter 5

Happy Coloring Problems

In a vertex-colored graph, an edge is *happy* if its endpoints have the same color. Similarly, a vertex is *happy* if all its incident edges are happy. Alternatively, a vertex is happy if it and all its neighbors have the same color. Given a partial coloring of the vertices of the graph using k colors, the MAXIMUM HAPPY VERTICES (also called k -MHV) problem asks to color the remaining vertices such that the number of happy vertices is maximized. The MAXIMUM HAPPY EDGES (also called k -MHE) problem asks to color the remaining vertices such that the number of happy edges is maximized. For arbitrary graphs, k -MHV and k -MHE are NP-hard for $k \geq 3$.

In this chapter, we study the complexity of k -MHV and k -MHE problems for some special graph classes like trees, bipartite graphs, split graphs and complete graphs. We show that both k -MHV and k -MHE problems are polynomial-time solvable for trees and complete graphs and NP-hard for bipartite graphs and split graphs.

We also study the happy coloring problems from parameterized algorithms perspective. We show that the k -MHE problem admits a $(k + \ell)$ -kernel. We show that both WEIGHTED MHE and WEIGHTED MHV admits $O^*(2^n)$ time exact exponential time algorithm. We show that WEIGHTED MHE is polynomial-time solvable when the uncolored vertices induce a forest. By combining these with few simple reduction rules we show that k -MHE has $O^*(2^\ell)$ time algorithm and hence FPT with respect to the parameter ℓ , the number of happy edges. We also show that both k -MHE and k -MHV are in FPT with respect to the parameters tree-width and neighborhood diversity.

5.1 Algorithm for k -MHV Problem for Trees

We root the tree at an arbitrary vertex. Let T_v denotes the subtree rooted at a vertex v . Before presenting the algorithm we give a simple reduction rule, which can be executed in linear time.

Rule 1. *If a leaf vertex is uncolored, remove it and count the leaf vertex as happy.*

We can give the color of its parent to the uncolored leaf to make it happy. Hence, without loss of generality we can assume that all the leaves are colored.

We process the vertices of the rooted tree according to post order traversal. At each vertex v , we maintain a list of $2k$ integer values. The maximum value of these $2k$ values gives the maximum number of happy vertices in T_v , the sub tree rooted at v . The maximum value of the $2k$ values associated with the root gives us the maximum number of happy vertices of the tree. The corresponding optimal coloring can also be traced back in reverse direction. The list of $2k$ values defined as follows, for $1 \leq i \leq k$:

- $T_v[i, H]$: The maximum number of happy vertices in the subtree T_v , when v is colored i and is happy in T_v . That is, when v and all its children are colored i . Note that, here we focus on v being happy in the subtree T_v . The vertex v can become unhappy in the tree T because its parent gets another color.
- $T_v[i, U]$: The maximum number of happy vertices in T_v , when v is colored i and is unhappy in T_v . That is, when one or more children of v are colored with a color other than i .

Note that, if a vertex or some of its children are already colored, then some of the $2k$ values are invalid. We use -1 to denote an invalid value. We keep these $2k$ values in an array to access any specific item in constant time. The values are indexed in the order, $T_v[1, H], T_v[1, U], T_v[2, H], T_v[2, U], \dots, T_v[k, H], T_v[k, U]$.

The following expressions are defined to simplify some of the equations:

- $T_v[i, *]$: The maximum number of happy vertices in the subtree T_v , when v is colored i . v may be happy or unhappy. That is:

$$T_v[i, *] = \max\{T_v[i, H], T_v[i, U]\}. \quad (5.1)$$

- $T_v[i, -]$: The maximum number of happy vertices in T_v excluding v , when v is colored i .

$$T_v[i, -] = \max\{T_v[i, H] - 1, T_v[i, U]\}. \quad (5.2)$$

- $T_v[\bar{i}, *]$: The maximum number of happy vertices in the subtree T_v , when v is colored with color other than i .

$$T_v[\bar{i}, *] = \max_{r \neq i} \{T_v[r, *]\}. \quad (5.3)$$

- $T_v[\bar{i}, -]$: The maximum number of happy vertices in the subtree T_v excluding v , when v is colored with color other than i .

$$T_v[\bar{i}, -] = \max_{r \neq i} \{T_v[r, -]\}. \quad (5.4)$$

- $T_v[* , *]$: The maximum number of happy vertices in T_v . That is:

$$T_v[* , *] = \max\{T_v[1, *], T_v[2, *], \dots, T_v[k, *]\}. \quad (5.5)$$

Now we explain the process to compute these $2k$ values at each vertex. As a leaf vertex is pre-colored, it is always happy alone as a subtree with a single vertex. Only one out of $2k$ values is valid. Suppose the color of the leaf is i , then the only valid value is $T_v[i, H] = 1$.

The following subsections consider the case when v is a non leaf vertex. Let v_1, v_2, \dots, v_d be the children of v . The values $T_v[i, H]$ and $T_v[i, U]$ are invalid, if v is pre-colored with a color $r \neq i$. Otherwise, we compute $T_v[i, H]$ and $T_v[i, U]$ as follows:

5.1.1 Computing $T_v[i, H]$

Computing $T_v[i, H]$ has two cases:

Algorithm 1 Computing $T_v[i, H]$

```

1: procedure COMPUTETVH( $v, i$ )
2:   if  $\forall v_j, T_{v_j}[i, *] \neq -1$  then
3:     return  $(1 + \sum_{v_j} T_{v_j}[i, *])$  ▷ Case 2
4:   else
5:     return  $-1$  ▷ Case 1
6:   end if
7: end procedure

```

Case 1: For some child v_j , $T_{v_j}[i, *] = -1$.

This means that the child v_j is pre colored with a color other than i . In this case, v becomes unhappy when it gets color i . So $T_v[i, H]$ is invalid.

Case 2: For every child v_j , $T_{v_j}[i, *] > -1$.

In this case, we use the following equation to compute $T_v[i, H]$.

$$T_v[i, H] = 1 + \sum_{v_j} T_{v_j}[i, *]. \quad (5.6)$$

5.1.2 Computing $T_v[i, U]$

Computing $T_v[i, U]$ has three cases:

Algorithm 2 Computing $T_v[i, U]$

```

1: procedure COMPUTETVU( $v, i$ )
2:   if every child  $v_j$  is pre-colored with color  $i$  then
3:     return  $-1$  ▷ Case 1
4:   else if  $\exists v_{j'}$  child of  $v$  such that  $T_{v_{j'}}[*] \neq T_{v_{j'}}[i, *]$  then
5:     return  $(\sum_{v_j} \max\{T_{v_j}[1, -], \dots, T_{v_j}[i, *], \dots, T_{v_j}[k, -]\})$  ▷ Case 2
6:   else ▷ Case 3
7:     for each child  $v_j$  do
8:        $\text{diff}(v_j, i) \leftarrow T_{v_j}[i, *] - T_{v_j}[\bar{i}, -]$ 
9:     end for
10:     $v_\ell \leftarrow \text{argmin}_{v_j} \text{diff}(v_j, i)$ 
11:     $q \leftarrow \text{argmax}_{r \neq i} T_{v_\ell}[r, -]$ 
12:    return  $(T_{v_\ell}[q, -] + \sum_{v_j \neq v_\ell} T_{v_j}[i, *])$ 
13:  end if
14: end procedure

```

Case 1: Every child v_j is pre colored with color i .

In this case, we cannot make v unhappy by giving color i to v . Hence $T_v[i, U]$ is invalid.

Case 2: For some child $v_{j'}$, $T_{v_{j'}}[*] \neq T_{v_{j'}}[i, *]$.

That is, the child $v_{j'}$ has color $r \neq i$ in the optimal coloring of $T_{v_{j'}}$. When v is colored i and $v_{j'}$ is colored r , irrespective of the colors of the other children, v will certainly be unhappy. In this case, we use the following expression to compute $T_v[i, U]$.

$$T_v[i, U] = T_{v_{j'}}[r, -] + \sum_{\substack{v_j \text{ child of } v, \\ v_j \neq v_{j'}}} \max\{T_{v_j}[1, -], \dots, T_{v_j}[i, *], \dots, T_{v_j}[k, -]\} \quad (5.7)$$

$$= \sum_{v_j \text{ child of } v} \max\{T_{v_j}[1, -], \dots, T_{v_j}[i, *], \dots, T_{v_j}[k, -]\}. \quad (5.8)$$

Case 3: For every child v_j , $T_{v_j}[*] = T_{v_j}[i, *]$.

For each v_j , if we pick $T_{v_j}[i, *]$, v will become happy, but we need v to be unhappy. To avoid this situation, for some child we pick a value with color other than i as follows:

For each v_j , we define $\text{diff}(v_j, i)$ as follows:

$$\text{diff}(v_j, i) = T_{v_j}[i, *] - T_{v_j}[\bar{i}, -]. \quad (5.9)$$

We pick the child (say v_ℓ) with minimum $\text{diff}(v_j, i)$ value. Suppose, $T_{v_\ell}[\bar{i}, -] = T_{v_\ell}[q, -]$, we replace $T_{v_\ell}[i, *]$ with $T_{v_\ell}[q, -]$. The new expression is:

$$T_v[i, U] = T_{v_\ell}[q, -] + \sum_{v_j \neq v_\ell} T_{v_j}[i, *]. \quad (5.10)$$

Algorithm 3 Algorithm for k -MHV problem

```

1: for each  $v \in V(G)$  in post order do
2:   for  $i = 1$  to  $k$  do
3:     if  $v$  is a leaf then
4:       if  $\text{color}(v) = i$  then
5:          $T_v[i, H] \leftarrow 1$ 
6:          $T_v[i, U] \leftarrow -1$ 
7:       else
8:          $T_v[i, H] \leftarrow -1$ 
9:          $T_v[i, U] \leftarrow -1$ 
10:      end if
11:     else
12:       if  $v$  is pre-colored and  $\text{color}(v) \neq i$  then
13:          $T_v[i, H] \leftarrow -1$ 
14:          $T_v[i, U] \leftarrow -1$ 
15:       else
16:          $T_v[i, H] \leftarrow \text{COMPUTETvH}(v, i)$ 
17:          $T_v[i, U] \leftarrow \text{COMPUTETvU}(v, i)$ 
18:       end if
19:     end if
20:   end for
21: end for

```

Theorem 20. *There is an $O(nk \log k)$ time algorithm for the k -MHV problem for trees.*

Proof. We evaluate the time spent at a particular vertex v to compute $T_v[i, H]$ and $T_v[i, U]$, for $1 \leq i \leq k$. Let v_1, v_2, \dots, v_d be the children of v .

Computing $T_v[i, H]$: The $T_{v_j}[i, H]$ and $T_{v_j}[i, U]$ values are accessible in constant time for each child v_j . Time to compute $T_v[i, H]$, $\forall 1 \leq i \leq k$ is:

$$\sum_{1 \leq i \leq k} O(d) = O(kd). \quad (5.11)$$

Computing $T_v[i, U]$: We sort the $2k$ values in descending order. For any child v_j , $T_{v_j}[i, *]$ is available in constant time from the original array. From the sorted array $T_{v_j}[*]$ and $T_{v_j}[\bar{i}, *]$ are available in constant time. Hence $T_v[i, U]$, $\forall 1 \leq i \leq k$ can be computed in:

$$O(dk \log k) + \sum_{1 \leq i \leq k} O(d) = O(dk \log k). \quad (5.12)$$

Hence the total time is:

$$\sum_v dk + dk \log k \leq \sum_v 2dk \log k = 2k \log k \sum_v d = O(nk \log k). \quad (5.13)$$

□

The correctness of the value $T_v[*]$ for every vertex v implies the correctness of the algorithm. The correctness of the value $T_v[*]$ follows from the correctness of the $2k$ values $T_v[1, H]$, $T_v[1, U]$, $T_v[2, H]$, $T_v[2, U]$, \dots , $T_v[k, H]$, $T_v[k, U]$ associated with v .

Theorem 21. *Algorithm 3 correctly computes the values $T_v[i, H]$ and $T_v[i, U]$ for every v and $1 \leq i \leq k$.*

Proof. We prove the theorem by using induction on the size of the subtrees. For a leaf vertex v , the algorithm correctly computes the values $T_v[i, H]$ and $T_v[i, U]$ for $1 \leq i \leq k$. Since the leaf vertices are pre-colored, each leaf vertex has only one valid value (this value being 1).

For a non-leaf vertex v , let v_1, v_2, \dots, v_d be the children of v . By induction on the size of the sub-trees, all the $2k$ values associated with each child v_j of v are correctly computed. Let x be the value computed by the algorithm for $T_v[i, H]$ (or $T_v[i, U]$) for any color i . If x is not the optimal value, it will contradict the optimality of at least one value of a child of v . Hence the algorithm correctly computes the values $T_v[i, H]$ and $T_v[i, U]$ for every v and $1 \leq i \leq k$. □

5.1.3 Generating all optimal happy vertex colorings

Our algorithm can also be extended to generate all the optimal happy vertex colorings of the tree. Among the $2k$ values associated with a vertex v , there may be multiple values equal to the optimal value. So, while generating optimal happy vertex coloring, we can choose any of these values to generate a different optimal coloring. For example, let $T_v[i, H]$ be an optimal value for the vertex v . Let v_j be a child of v with both $T_{v_j}[i, H]$ and $T_{v_j}[i, U]$ are optimal. So, we can generate one optimal coloring by picking $T_{v_j}[i, H]$ and another optimal coloring by picking $T_{v_j}[i, U]$. There may be exponentially many optimal colorings, but, generating each optimal coloring takes polynomial-time (linear time for fixed k).

5.2 Algorithm for k -MHE problem for Trees

Before presenting the algorithm we give simple reduction rules, which can be executed in linear time.

Rule 2. *Let v be a pre-colored vertex with degree more than 1. Let v_1, v_2, \dots, v_d be the neighbours of v in T . We can divide T into d edge disjoint subtrees T_1, T_2, \dots, T_d and all these trees share only the vertex v .*

$$k\text{-MHE}(T) = k\text{-MHE}(T_1) + k\text{-MHE}(T_2) + \dots + k\text{-MHE}(T_d). \quad (5.14)$$

With the application of Rule 2, without loss of generality we can assume that T does not have a pre-colored vertex with degree more than 1.

Now, we root the tree at an arbitrary vertex with degree more than 1.

Rule 3. *(Similar to Rule 1 in Section 5.1) If a leaf vertex is uncolored, remove it and count the edge connecting the leaf vertex as happy.*

With Rule 2 and Rule 3, without loss of generality, all the leaves of the rooted tree T are pre-colored and no non-leaf vertex is pre-colored.

Our algorithm for k -MHE problem has two phases. In the first phase, we visit the vertices according to post order traversal and populate a list of tentative colors for each vertex. In the second phase we visit the vertices according to pre-order traversal and assign a color for each vertex.

Phase 1: We visit the vertices according to post order traversal. At each vertex v , we keep a list of tentative colors to assign to the vertex v in the optimal solution. The size of this list is at most k . Let $L(v)$ denote the list of tentative colors associated with the vertex v .

If the vertex v is a leaf, as the leaf vertex is pre-colored, we add that pre-color to $L(v)$. Otherwise, let v_1, v_2, \dots, v_d be the children of v . The list of tentative colors $L(v_j)$ for each vertex v_j are already computed. For each child v_j , we traverse the list $L(v_j)$ and compute the frequency of occurrences of each color in the multiset that is union of the lists. Let $\text{frequency}(i)$ denote the frequency of color i . We add all the colors with maximum frequency to $L(v)$. The process is captured in Algorithm 4.

Phase 2: We visit the vertices according to pre-order traversal to assign a color to each vertex. Let v be the vertex in pre-order. If $|L(v)| = 1$, then we fix the color of v to the only color in $L(v)$. Otherwise, we check if the color of the parent of v is present in $L(v)$, and assign it to v if present. Otherwise, we pick any arbitrary color from $L(v)$ and assign it to v . The process is captured in Algorithm 5.

Algorithm 4 Phase 1 of the algorithm

```
1: procedure POPULATE TENTATIVE COLORS( $T$ )
2:   for each  $v \in V(G)$  in post order do
3:     if  $v$  is a leaf then
4:        $L(v) \leftarrow \text{color}(v)$ 
5:     else ▷ Let  $v_1, v_2, \dots, v_d$  be the children of  $v$ 
6:       frequency[1.. $k$ ]  $\leftarrow \{0\}$ 
7:       for each child  $v_j$  of  $v$  do
8:         for each color  $c \in L(v)$  do
9:           frequency[ $c$ ]  $\leftarrow$  frequency[ $c$ ] + 1
10:        end for
11:       end for
12:       max  $\leftarrow 0$ 
13:       for  $i = 1$  to  $k$  do
14:         if frequency[ $i$ ] > max then
15:           max  $\leftarrow$  frequency[ $i$ ]
16:         end if
17:       end for
18:       for  $i = 1$  to  $k$  do
19:         if frequency[ $i$ ] = max then
20:            $L(v) \leftarrow L(v) \cup \{i\}$ 
21:         end if
22:       end for
23:     end if
24:   end for
25: end procedure
```

Theorem 22. *There is an $O(nk)$ time algorithm for the k -MHE problem for trees.*

Proof. At each vertex with degree d , we perform $O(kd)$ time in the Phase 1 and $O(k)$ time in the Phase 2. The time complexity is:

$$\sum_v O(kd) = O(nk). \quad (5.15)$$

□

The correctness of the algorithm can be proved using induction on the size of the sub-tree similar to Theorem 21.

Algorithm 5 Phase 2 of the algorithm

```
1: procedure ATTACHCOLORS( $T, L$ ) ▷ Fixing color to vertices
2:   for each  $v \in V(G)$  in pre order do
3:     if  $|L(v)| = 1$  then
4:        $\text{color}(v) \leftarrow$  Only element of  $L(v)$ 
5:     else if  $\text{color}(\text{parent}(v)) \in L(v)$  then
6:        $\text{color}(v) \leftarrow \text{color}(\text{parent}(v))$ 
7:     else
8:        $\text{color}(v) \leftarrow$  Any element of  $L(v)$ 
9:     end if
10:  end for
11: end procedure
```

5.2.1 Generating all optimal happy edge colorings

Our algorithm can be extended to generate all the optimal happy edge colorings. We keep a list of tentative colors at each vertex. At a vertex v , if the $\text{color}(\text{parent}(v))$ is present in $L(v)$, then, we assign the $\text{color}(\text{parent}(v))$ to v in the optimal coloring. Otherwise, we can generate a different optimal coloring for each color in $L(v)$. Here we point out that, this scheme may miss out some optimal colorings when $\text{color}(\text{parent}(v))$ is not present in $L(v)$ but present in the set of colors with frequency one less than the maximum frequency. In this case, we can assign the $\text{color}(\text{parent}(v))$ to v even though the $\text{color}(\text{parent}(v))$ is not present in $L(v)$. A special case of this scenario is when there is a vertex v where all its children have distinct colors (the maximum frequency being 1). Even though the $\text{color}(\text{parent}(v))$ not present in $L(v)$, we can assign the $\text{color}(\text{parent}(v))$ to v as it has zero frequency at v .

There may be exponentially many optimal happy edge colorings. Generating each optimal coloring takes polynomial-time (linear time for fixed k).

5.3 MHV and MHE on Complete Graphs

MHV problem is trivial on complete graphs. Thus we have the following proposition.

Proposition 23. *Any partial coloring c of the complete graph K_n for any $n \geq 1$ can be extended to a full coloring c' making n vertices happy iff c uses at most one color. Consequently, the problem MHV is solvable in polynomial-time for complete graphs.*

Proposition 24. *The problem MHE is solvable in polynomial-time for complete graphs.*

Proof. Let S denote the set of precolored vertices for the K_n for any $n \geq 1$. Delete edges whose both endpoints are in S , since their happiness is already determined by the

precoloring. Observe that S is now an independent set and $C = V(G) \setminus S$ induces a clique. Moreover, every vertex in S is adjacent to every vertex in C .

Denote by p the most frequent occurrence of any color among the precolored vertices. For any vertex $v \in C$, regardless of the color we give to v , we can make at most p edges happy among the edges from the vertices in S to v . Thus, the number of happy edges is at most $p \cdot |C| + |E(C)|$. In fact, we can achieve exactly $p \cdot |C| + |E(C)|$ happy edges by giving a single color to all the vertices in C . More precisely, we color all the uncolored vertices with the color that is used p times, completing the proof. \square

We remark that for the above proof to hold, we do not need the graph to be complete. Indeed, the procedure described in the proof can be applied as long as every precolored vertex is adjacent to every uncolored vertex.

5.4 Hardness Results for Happy Coloring Problems

We begin this section by proving hardness of both DMHE and DMHV for bipartite graphs and split graphs. To prove the NP-hardness we consider the following decision versions of MHE and MHV.

DMHV

Parameter: ℓ

Input: A graph G , integers k and ℓ , a vertex subset $S \subseteq V(G)$, (partial) coloring $c : S \rightarrow [k]$.

Question: Does there exist a coloring $\tilde{c} : V(G) \rightarrow [k]$ such that $\tilde{c}|_S = c$ and the number of happy vertices is at least ℓ ?

DMHE

Parameter: ℓ

Input: A graph G , integers k and ℓ , a vertex subset $S \subseteq V(G)$, (partial) coloring $c : S \rightarrow [k]$.

Question: Does there exist a coloring $\tilde{c} : V(G) \rightarrow [k]$ such that $\tilde{c}|_S = c$ and the number of happy edges is at least ℓ ?

Theorem 25. *The DMHV problem is NP-complete for split graphs.*

Proof. Let $I = (G, c, \ell)$ be an instance of DMHE, and let us in polynomial-time construct an instance $I' = (G', c', \ell)$ of DMHV. We can safely (and crucially) assume at least two vertices of G are precolored (in distinct colors), for otherwise the instance is trivial. We construct the split graph $G' = (C \cup B, E' \cup E'')$, where

- $C = \{v_x \mid x \in V(G)\}$,
- $B = \{v_e \mid e \in E(G)\}$,

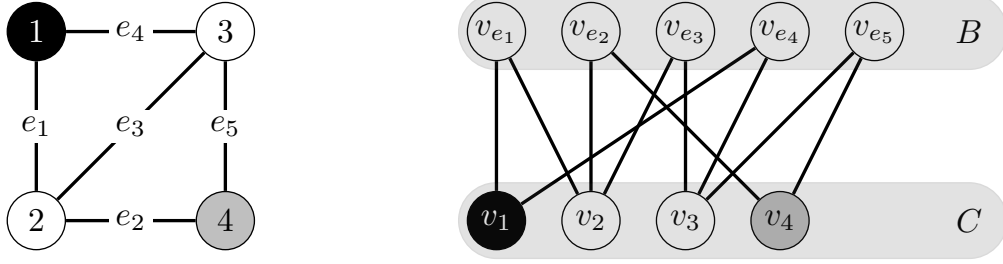


Figure 5.1: **(a)** A graph G of an instance of DMHE, where white vertices correspond to uncolored vertices. **(b)** The graph G transformed into a split graph G' by the construction of Theorem 25. The edges between the vertices in C are not drawn.

- $E' = \{v_e v_x \mid e \text{ is incident to } x \text{ in } G\}$, and
- $E'' = \{v_x v_{x'} \mid x, x' \in V(G)\}$.

That is, C forms a clique and B an independent set in G' , proving G' is split. In particular, observe that the degree of each vertex v_e is two. To complete the construction, we retain the precoloring, i.e., set $c'(v_x) = c(x)$ for every $x \in V(G)$. The construction is illustrated in Figure 5.1.

We claim that I is a YES-instance of DMHE iff I' is a YES-instance of DMHV. Suppose ℓ edges can be made happy in G by an extended full coloring of c . Consider an edge $e \in E(G)$ whose endpoints are colored with color i . To make ℓ vertices happy in G' , we give v_e and its two neighbors the color i . For the other direction, suppose ℓ vertices are happy under an extended full coloring of c' . As at least two vertices in C are colored in distinct colors, it follows by Proposition 23 that all the happy vertices must be in B . Furthermore, the vertices in B correspond to precisely the edges in $E(G)$, so we are done. \square

Theorem 26. *The DMHV problem is NP-complete for bipartite graphs.*

Proof. We start with the construction of Theorem 25. Modify the split graph G' by deleting the edges between the vertices in C , i.e., let $G'' = (C \cup B, E')$. For each $v_x \in C$, add a path $S_{v_x} = \{v_x^1, v_x^2, v_x^3\}$ along with the edges $v_x v_x^1$ and $v_x^3 v_x$. In other words, each v_x forms a 4-cycle with the vertices in S_{v_x} . Clearly, we have that G'' is bipartite as it contains no odd cycles. Arbitrarily choose three distinct colors from $[k]$, and map them bijectively to S_{v_x} . Observe that by construction, none of the vertices in S_{v_x} can be happy under any c' extending c . This completes the construction. Correctness follows by the same argument as in Theorem 25. \square

Theorem 27. *The DMHE problem is NP-complete for bipartite graphs.*

Table 5.1: Summary of our hardness results for happy coloring problems

Graph class	k -MHE	k -MHV
Bipartite	NPC	NPC
Complete	P	P
Split	NPC [66]	NPC

Proof. Let $I = (G, c, \ell)$ be an instance of DMHE, and let us in polynomial-time construct an instance $I' = (G', c, m + \ell)$ of DMHE, where G' is bipartite. We obtain G' by subdividing every edge of G . Observe that if G has n vertices and m edges, then G' has $n + m$ vertices and $2m$ edges. Clearly, G' is bipartite.

We will now show that G has an extended full coloring making at least ℓ edges happy iff G' has an extended full coloring making at least $m + \ell$ edges happy. Let c' be an extended full coloring of the precoloring c given to G . We give G' the same extended full coloring, and give each vertex in $v \in V(G') \setminus V(G)$ an arbitrary color that appears on a vertex adjacent to v . Thus, for each edge in G , we have one extra happy edge in G' , giving us a total of at least $m + \ell$ happy edges. For the other direction, let c' be an extended full coloring of c that makes at least $m + \ell$ edges happy in G' . Now, there are at least ℓ vertices in $V(G') \setminus V(G)$ with both of its incident edges happy. These 2ℓ happy edges correspond to the ℓ happy edges in G . This concludes the proof. \square

We were not able to prove the NP-hardness of DMHE for split graphs. However, Mishra and Reddy [66] gave the proof for the NP-hardness of DMHE for split graphs.

5.4.1 k -MHE for planar graphs and graphs with bounded branch width

The MULTIWAY CUT is NP-hard for planar graphs [26] when k , the number of terminals, is not fixed. This implies the following theorem on hardness of k -MHE for planar graphs for an arbitrary k .

Theorem 28. *For an arbitrary k , the k -MHE problem is NP-hard for planar graphs.*

In [72], Robertson and Seymour introduced the notions of tree width and branch width. They showed that these two quantities are always within a constant factor of each other. Many graph problems that are NP-hard for general graphs have been shown to be solvable in polynomial time for graphs with bounded tree width or equivalently bounded branch width.

Definition 35. MULTI-MULTIWAY CUT (*Instance*) We are given an undirected graph $G = (V(G), E(G))$ and c sets of vertices S_1, S_2, \dots, S_c .

(Goal) Find a set of edges $C \subseteq E(G)$ with minimum cardinality whose removal disconnects every pair of vertices in each set S_i .

When $c = 1$, the MULTI-MULTIWAY CUT problem is equivalent to MULTIWAY CUT problem. The k -MHE problem can also be formulated as a MULTI-MULTIWAY CUT problem, by creating vertex sets with every pair of pre-colored vertices with different colors. In [27], Deng et. al. studied the MULTI-MULTIWAY CUT problem for graphs with bounded branch width and presented an $O(b^{2b+2} \cdot 2^{2bc} \cdot |G|)$ time algorithm, where b is the branch width of the graph and c is the number of vertex sets. The algorithm runs in linear time when the branch width and the number of vertex sets are fixed.

Theorem 29. *When the branch width of the graph and the number of pre-colored vertices are bounded, there is a linear time algorithm for the k -MHE problem.*

Proof. Let the number of pre-colored vertices be p and the branch width be b . For this instance of k -MHE, we can formulate a MULTI-MULTIWAY CUT problem with at most p^2 vertex sets. Hence, the k -MHE problem can be solved in time $O(b^{2b+2} \cdot 2^{2bp^2} \cdot |G|)$. Hence, when both the number of pre-colored vertices and the branch width are constants, the k -MHE problem can be solved in linear time. \square

5.5 Exact Exponential-Time Algorithms for Happy Coloring

In this section, we consider the happy coloring problems from the viewpoint of exact exponential-time algorithms. Every problem in NP can be solved in time exponential in the input size by a brute-force algorithm. For WEIGHTED MHE (WEIGHTED MHV), such an algorithm goes through each of the at most k^n colorings, and outputs the one maximizing the total weight of the happy edges (vertices). It is natural to ask whether there is an algorithm that is considerably faster than the $k^n n^{O(1)}$ -time brute force approach. In what follows, we show that brute-force can be beaten. Let us introduce the following more general problem.

MAX WEIGHTED PARTITION

Input: An n -element set N , integer d , and functions $f_1, f_2, \dots, f_d : 2^N \rightarrow [-M, M]$ for some integer M .

Question: A d -partition (S_1, S_2, \dots, S_d) of N that maximizes $f_1(S_1) + f_2(S_2) + \dots + f_d(S_d)$.

Using an algebraic approach, the following has been shown regarding the complexity of the problem.

Theorem 30 (Björklund, Husfeldt, Koivisto [10]). *The MAX WEIGHTED PARTITION problem can be solved in $3^n d^2 M \cdot n^{O(1)}$ time and polynomial space. In exponential space, the time can be improved to $2^n d^2 M \cdot n^{O(1)}$.*

In the following, we observe that the weighted variants of both problems can be reduced to MAX WEIGHTED PARTITION. This results in an algorithm that is considerably faster than one running in time $k^n n^{O(1)}$.

Lemma 31. *WEIGHTED MHE and WEIGHTED MHV reduce in polynomial-time to MAX WEIGHTED PARTITION.*

Proof. Consider the claim for an instance $I = (G, w, k, S, c)$ of WEIGHTED MHE. To construct an instance of MAX WEIGHTED PARTITION, let $N = V(G) \setminus S$, where S is the set of precolored vertices, let $d = k$, and let $M = \sum_{uv \in E(G)} w(uv)$. Define $f_i = \sum_{uv \in E(G[S_i \cup c^{-1}(i)])} w(uv)$, i.e., f_i sums the weights of the edges uv that range over the edge set of the subgraph induced by the union of S_i and $c^{-1}(i)$, the vertices precolored with color i . Thus, a partition (S_1, \dots, S_k) maximizing $f_1(S_1) + \dots + f_k(S_k)$ maximizes the weight of happy edges.

Finally, consider the claim for an instance $I = (G, w, k, S, c)$ of WEIGHTED MHV. Now, we define $f_i = \sum_{v \in S_i: \forall y \in N(v): y \in (S_i \cup c^{-1}(i))} w(v)$, i.e., f_i sums the weights of the vertices v for which it holds that v and each neighbor y of v are all colored with color i . Also, we let $M = \sum_{v \in V(G)} w(v)$, but otherwise the argument is the same as above. \square

For some NP-complete problems, the fastest known algorithms run in $O^*(2^n)$ time, but we do not necessarily know whether (under reasonable complexity-theoretic assumptions) they are optimal. Indeed, could one have an algorithm that runs in $O^*((2 - \varepsilon)^n)$ time, for any $\varepsilon > 0$, for either WEIGHTED MHE or WEIGHTED MHV? We prove that at least for some values of k this bound can be achieved. For this, we recall the following result.

Theorem 32 (Zhang and Li [78]). *For $k = 2$, k -MHE and k -MHV are solvable in $O(\min\{n^{2/3}m, m^{3/2}\})$ and $O(mn^7 \log n)$ time, respectively.*

We are ready to proceed with the following.

Lemma 33. *For $k = 3$, k -MHE and k -MHV can be solved in time $O^*(1.89^{n'})$, where n' is the number of uncolored vertices in the input graph.*

Proof. First, consider the claim for an instance $I = (G, S, c)$ of k -MHE. Consider a partition $\mathcal{S} = (S_1, S_2, S_3)$ of the uncolored vertices into $k = 3$ color classes that maximizes the number of happy edges. Also, denote by C_i for $i \in [k]$ the set of vertices precolored with color i . In $V(G) \setminus (S_3 \cup C_3)$, by the optimality of \mathcal{S} , it must be the case that

$S_1 \cup C_1$ and $S_2 \cup C_2$ have a minimum number of crossing edges. Thus, we can proceed as follows. Observe that in any optimal solution \mathcal{S} , there exists $S_i \in \mathcal{S}$ such that $|S_i| \leq n'/3$. The number of subsets of size at most $n'/3$ is $2^{H(1/3)n'} < 1.89^{n'}$, using the well-known bound $2^{H(1/3)} < 1.89$, where $H(\cdot)$ is the binary entropy function (for a proof, see e.g., [34, Lemma 3.13]). Thus, we guess S_i by extending it in all possible at most $1.89^{n'}$ ways. Then, for every such partial coloring, we solve an instance of 2-MHE on the remaining graph $G[V(G) \setminus (S_1 \cup C_1)]$ in polynomial-time by Theorem 32. Combining the bounds, we obtain an algorithm running in time $O^*(1.89^{n'})$ for 3-MHE.

The observation is similar for 3-MHV, but we solve an instance of 2-MHV on $V(G) \setminus N[S_i \cup C_i]$ instead of $V(G) \setminus (S_i \cup C_i)$. \square

By Lemma 33 and by combining Theorem 30 with Lemma 31, we arrive at the following.

Theorem 34. *For every $k \geq 3$, WEIGHTED k -MHE and WEIGHTED k -MHV can be solved in time $O^*(2^{n'})$. When $k = 3$, the problems k -MHE and k -MHV are solvable in time $O^*(1.89^{n'})$, where n' is the number of uncolored vertices in the input graph.*

5.6 A Linear Kernel for WEIGHTED MHE

In this Section we prove that WEIGHTED DMHE has a kernel of size $k + \ell$. Our strategy to obtain the kernel consists of two parts: first, we will show that there is a polynomial-time algorithm for WEIGHTED MHE when the uncolored vertices induce a forest. Then, to leverage this algorithm, we apply a set of reduction rules that shrink the instance considerably, or solve it directly along the way.

5.6.1 Polynomial Time Algorithm for Subproblems of WEIGHTED MHE

We show that the WEIGHTED MHE problem is polynomial-time solvable when the uncolored vertices $V(G) \setminus S$ induce a tree, where S is the set of precolored vertices. When $V(G) \setminus S$ induces a forest, we run the algorithm for each component in $V(G) \setminus S$ independently. The approach we present is based on dynamic programming, and inspired by the algorithm given in [4].

We define edges *touching* a subtree to be those edges that have at least one endpoint in the subtree. We choose any vertex $r \in V(G) \setminus S$ as the root of the tree induced by $V(G) \setminus S$. The vertices of this rooted tree are processed according to its post-order traversal. At each node, we keep k values. The k values are defined as follows, for $1 \leq i \leq k$:

- $T_v[i]$: The maximum total weight of the happy edges touching the subtree T_v , when the vertex v is colored with color i .

We also define the following expressions:

- $T_v[*]$: The maximum total weight of the happy edges touching the subtree T_v , i.e.,

$$T_v[*] = \max_{i=1}^k \{T_v[i]\}. \quad (5.16)$$

- $T_v[\bar{i}]$: The maximum total weight of the happy edges touching the subtree T_v , when the vertex v is colored with a color other than i , i.e.,

$$T_v[\bar{i}] = \max_{j=1, j \neq i}^k \{T_v[j]\}. \quad (5.17)$$

If W_p is the total weight of the happy edges in the initial partial coloring, $W_p + T_r[*]$ gives us the maximum total weight of the happy edges in G . Now, we explain how to compute the values $T_v[i]$ for $1 \leq i \leq k$ and for each $v \in V(G) \setminus S$. When we say *color- i* vertices, we mean the vertices precolored with color i .

For a leaf vertex $v \in V(G) \setminus S$, let v_1, v_2, \dots, v_x be the color- i neighbors of v in G . Then,

$$T_v[i] = \sum_{j=1}^x w(vv_j). \quad (5.18)$$

If there are no color- i neighbors for v , then $T_v[i]$ is set to 0.

For a non-leaf vertex $v \in V(G) \setminus S$, let v_1, v_2, \dots, v_x be the color- i neighbors of v in G and let u_1, u_2, \dots, u_d be the children of v in $V(G) \setminus S$. Then,

$$T_v[i] = \sum_{j=1}^x w(vv_j) + \sum_{j=1}^d \max\{(w(vu_j) + T_{u_j}[i]), T_{u_j}[\bar{i}]\}. \quad (5.19)$$

This naturally leads to an algorithm listed as Algorithm 6.

The running time of the algorithm is $O(k(m+n))$. The correctness of the values $T_v[i]$, for $1 \leq i \leq k$ and for each $v \in V(G) \setminus S$, implies the correctness of the algorithm. The following theorem is proved by induction on the size of the subtrees.

Theorem 35. *Algorithm 6 correctly computes the values $T_v[i]$ for every $v \in V(G) \setminus S$ and $1 \leq i \leq k$.*

Proof. We prove the theorem by using induction on the size of the subtrees. For a leaf vertex v , the algorithm correctly computes the values $T_v[i]$ for $1 \leq i \leq k$. For a non-leaf

Algorithm 6 Algorithm for a special case of WEIGHTED MHE

Input: A weighted undirected graph G with $S \subseteq V(G)$ precolored vertices under a partial vertex-coloring $c : S \rightarrow [k]$, $V(G) \setminus S$ induces a tree, and a vertex $r \in V(G) \setminus S$ as the root of the tree.

Output: Maximum total weight of the happy edges in G .

```
1:  $M_p \leftarrow 0$ 
2: for each happy edge  $uv$  in the precoloring do
3:    $M_p \leftarrow M_p + w(uv)$ 
4: end for
5: for each  $v \in V(G) \setminus S$  in post-order do
6:   if  $v$  is a leaf vertex in  $V(G) \setminus S$  then
7:     for  $i = 1$  to  $k$  do
8:        $T_v[i] \leftarrow 0$ 
9:       for each  $vu \in E(G)$  such that  $u \in S$  and  $c(u) = i$  do
10:         $T_v[i] \leftarrow T_v[i] + w(vu)$ 
11:      end for
12:    end for
13:   else
14:     for  $i = 1$  to  $k$  do
15:        $T_v[i] \leftarrow 0$ 
16:       for each  $vu \in E(G)$  such that  $u \in S$  and  $c(u) = i$  do
17:         $T_v[i] \leftarrow T_v[i] + w(vu)$ 
18:      end for
19:       for each child  $u$  of  $v$  in  $V(G) \setminus S$  do
20:         $T_v[i] \leftarrow T_v[i] + \max\{w(vu) + T_u[i], T_u[\bar{i}]\}$ 
21:      end for
22:     end for
23:   end if
24: end for
25: return  $(M_p + T_r[*])$ 
```

vertex v , let u_1, u_2, \dots, u_d be the children of v in $V(G) \setminus S$. By induction, all the k values associated with each child u_j of v are correctly computed. Moreover, $T_v[i]$ is the sum of two quantities (see Equation 5.19), the first quantity is correct because it is the sum of the weights of the happy edges from v to S . If $T_v[i]$ is not correct, it will contradict the correctness of $T_{u_j}[*]$ for some child u_j of v . So, the second term in the $T_v[i]$ is correct. Hence, the algorithm correctly computes the values $T_v[i]$ for every v in $V(G) \setminus S$ and $1 \leq i \leq k$. \square

In this subsection, we assume the edge weights of the WEIGHTED DMHE instance are positive integers. The kernel will also work for real weights that are at least 1. We present the following simple reduction rules.

Rule 4. *If G contains an isolated vertex, delete it.*

Rule 5. *If both endpoints of an edge $uv \in E(G)$ are colored, remove uv . Furthermore, if $c(u) = c(v)$, decrement ℓ by the weight on uv .*

Proof. As both endpoints of uv are colored, the existence of the edge uv does not further contribute to the value of the optimal solution. Moreover, if the edge is already happy under c , we can safely decrement ℓ . \square

Rule 6. *Contract every color class C_i induced by the partial coloring c into a single vertex. Let e_1, \dots, e_r be the (parallel) edges between two vertices u and v . Delete each edge in e_1, \dots, e_r except for e_1 , and update $w(e_1) = w(e_1) + w(e_2) + \dots + w(e_r)$.*

Proof. Let G' be the resulting graph after the application of Rule 6. Because Rule 5 does not apply, each color class C_i forms an independent set. Thus, G' contains no self-loops.

Fix a color i , and consider an uncolored vertex $v \in V(G) \setminus C_i$. Denote by $N_i(v)$ the neighbors of v with color i , and denote by $E[X, Y]$ the set of edges whose one endpoint is in X and the other in Y . Depending on the color v gets in an extended full coloring of c , either all edges in $E[\{v\}, N_i(v)]$ are happy or all are unhappy. Hence, we can safely replace these edges with a single weighted edge. \square

Theorem 36. *The problem WEIGHTED DMHE admits a kernel on $k + \ell$ vertices.*

Proof. Let (G, w, k, S, c) be a reduced instance of WEIGHTED DMHE. We claim that if G has more than $k + \ell$ vertices, then we have YES-instance. The proof follows by the claims below.

Claim 1. *The weight of each edge is at most ℓ .*

Proof. If an edge uv has $w(uv) \geq \ell$ and at least one of u and v is uncolored, we make uv happy and output YES. On the other hand, any unhappy edge (with any weight) has been removed by Rule 5. \square

Claim 2. *The number of precolored vertices in G is at most k .*

Proof. Follows directly from Rule 6. \square

Claim 3. *The number of uncolored vertices in G is at most $\ell - 1$.*

Proof. Let H be the graph induced by the uncolored vertices, i.e., $H = G[V(G) \setminus \cup_{i \in [k]} C_i]$. We note the following two cases:

- If any of the connected components of H is a tree, then we apply the procedure described in Section 5.6.1 for that component, and decrement the parameter ℓ accordingly.

- If $w(E(H)) \geq \ell$, then we color all the vertices in H by the same color making all the edges in H happy. So the case where $w(E(H)) \geq \ell$ is a YES-instance.

After the application of the above, every component of H contains a cycle, and $|E(H)| < \ell$. So in each component of H , the number of vertices is at most the number of edges. Consequently, we have $|V(H)| \leq |E(H)| < \ell$. Hence the number of uncolored vertices is at most $\ell - 1$. \square

Clearly, all of the mentioned rules can be implemented to run in polynomial-time. Moreover, as we have bounded the number of precolored and uncolored vertices, the claimed kernel follows. \square

By combining Theorem 36 with Theorem 34, we have the following corollary.

Corollary 1. *The WEIGHTED DMHE problem can be solved in time $O^*(2^\ell)$. For the special case of $k = 3$, the problem WEIGHTED DMHE admits an algorithm running in time $O^*(1.89^\ell)$.*

5.7 Structural parameterization

In this section, we consider happy coloring from the standpoint of various structural parameters: tree-width and neighborhood diversity. The algorithms for tree-width were obtained independently by [66] and [3].

5.7.1 Tree-Width

Theorem 37. *For any $k \geq 1$, both WEIGHTED k -MHE and WEIGHTED k -MHV can be solved in time $k^t \cdot n^{O(1)}$, where n is the number of vertices of the input graph and t is its tree-width.*

Proof. Let us prove the statement for WEIGHTED k -MHE, and then explain how the proof extends for WEIGHTED k -MHV. Let (G, c, w, ℓ) be an instance of WEIGHTED k -MHE, let $(\{X_i \mid i \in I\}, T = (I, F))$ be a nice tree decomposition of G of width t , and let r be the root of T . Moreover, denote by G_i the subgraph of G induced by $\bigcup_j X_j$ where j belongs to the subtree of T rooted at i .

For every node i of T we set up a table K_i indexed by all possible extended full k -colorings of X_i . Intuitively, an entry of K_i indexed by $f : X_i \rightarrow [k]$ gives the total weight of edges happy in G_i under f . It holds that an optimal solution is given by $\max_f \{K_r[f]\}$. In what follows, we detail the construction of the tables K_i for every node i . The algorithm processes the nodes of T in a post-order manner, so when processing i , a table has been computed for all children of i .

- **Leaf node.** Let i be a leaf node and $X_i = \{v\}$. Obviously, G_i is edge-free, so we have $K_i[f] = 0$. As k is fixed, K_i is computed in constant time.
- **Introduce node.** Let i be an introduce node with child j such that $X_i = X_j \cup \{v\}$. Put differently, G_i is formed from G_j by adding v and a number of edges from v to vertices in X_j . The properties of a tree decomposition guarantee that $v \notin V(G_j)$, and that v is not adjacent to a vertex in $V(G_j) \setminus X_j$. It is not difficult to see that we set $K_i[f] = K_i[f|_{X_j}] + \sum_{p \in N_h(v)} w(pv)$, where $N_h(v)$ denotes the neighbors of v colored with the same color as v . It follows K_i can be computed in time $O(k^{t+1})$.
- **Forget node.** Let i be a forget node with child j such that $X_i = X_j \setminus \{v\}$. Observe that the graphs G_i and G_j are the same. Thus, we set $K_i[f]$ to the maximum of $K_j[f']$ where $f'|_{X_i} = f$. Since there are at most k such colorings f' for each f , we compute K_i in time $O(k^{t+2})$.
- **Join node.** Let i be a join node with children j_1 and j_2 such that $X_i = X_{j_1} = X_{j_2}$. The properties of a tree decomposition guarantee that $V(G_{j_1}) \cap V(G_{j_2}) = X_i$, and that no vertex in $V(G_{j_1}) \setminus X_i$ is adjacent to a vertex in $V(G_{j_2}) \setminus X_i$. Thus, we add together weights of happy edges that appear in G_{j_1} and G_{j_2} , while subtracting a term guaranteeing we do not add weights of edges that are happy in both subgraphs. Indeed, we set $K_i[f] = K_{j_1}[f] + K_{j_2}[f] - q$, where q is the total weight of the edges made happy under f in X_i . The table $K_i[f]$ can also be computed in time $O(k^{t+2})$.

To summarize, each table K_i has size bounded by k^{t+1} . Moreover, as each table is computed in $O(k^{t+2})$ time, the algorithm runs in $k^t \cdot n^{O(1)}$ time, which is what we wanted to show.

The proof is similar for WEIGHTED k -MHV, but each table now stores the total weight of the happy vertices under an extended full k -coloring. \square

5.7.2 Neighborhood Diversity

We proceed to present algorithms for MHE and MHV for graphs of bounded neighborhood diversity. Consider a type partition of a graph G with d sets, and an instance of $I = (G, k, S, c)$ of MHE (MHV). If a set contains both precolored and uncolored vertices, we split the set into two sets: one containing precisely the precolored vertices and the other precisely the uncolored vertices. After splitting each set, the number of sets is at most $2d$. For convenience, we say a set is *uncolored* if each vertex in it is uncolored; otherwise the set is *precolored*. Let the uncolored sets be P_1, P_2, \dots, P_d . In what follows, we discuss how vertices in these sets are colored in an optimal solution. We say a set is *monochromatic* if all of its vertices have the same color.

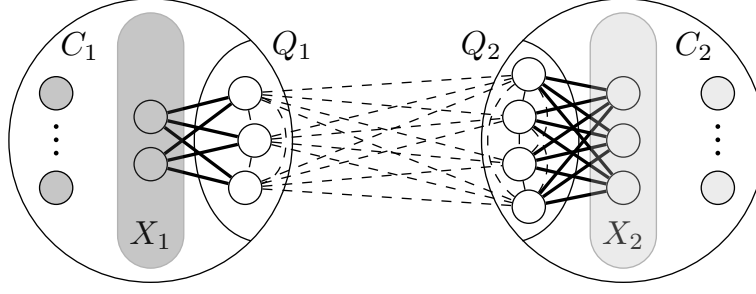


Figure 5.2: A set of a type partition, where each vertex in $Q_1 \cup Q_2$ has the same type. The dashed edges appear exactly when $Q_1 \cup Q_2$ induces a clique. The set Q_1 forms a complete bipartite graph with both X_1 and X_2 ; likewise for Q_2 (edges omitted for brevity).

MHE

Parameter: neighborhood diversity t

Input: A graph G , an integer k , a vertex subset $S \subseteq V(G)$, and a (partial) coloring $c : S \subseteq V(G) \rightarrow [k]$.

Output: A coloring $\tilde{c} : V(G) \rightarrow [k]$ such that $\tilde{c}|_S = c$ maximizing the number of happy edges.

Lemma 38. *There is an optimal extended full coloring for an instance I of MHE such that each uncolored set P_i for $1 \leq i \leq t$ is monochromatic.*

Proof. Consider any optimal extended full coloring for an instance I . Suppose the vertices in a set P_i belong to more than one color class. Let Q_1 and Q_2 be the (disjoint and non-empty) sets of vertices of P_i belonging to color classes C_1 and C_2 , respectively. Let X_1 and X_2 be the neighbors of the vertices in Q_1 and Q_2 in color classes C_1 and C_2 , respectively, as shown in Figure 5.2. Without loss of generality, let us assume that $|X_1| \leq |X_2|$. By recoloring vertices in Q_1 with the color of C_2 , we retain an optimal solution without disturbing the colors of other vertices. If $|E(Q_1, Q_2)|$ is the number of edges between Q_1 and Q_2 , the gain in the number of happy edges by recoloring Q_1 is $|E(Q_1, Q_2)| + |Q_1|(|X_2| - |X_1|)$, which is strictly positive if P_i is a clique and non-negative if P_i is an independent set.

In conclusion, we have shown that every optimal extended full coloring makes each P_i inducing a clique monochromatic. Moreover, there is an optimal extended full coloring making each P_i inducing an independent set monochromatic. \square

The previous lemma is combined with the algorithm of Theorem 34 to obtain the following.

Theorem 39. *For any $k \geq 1$, MHE can be solved in time $O^*(2^d)$, where d is the neighborhood diversity of the input graph.*

Proof. First we construct a weighted graph H from G as follows: merge each uncolored set into a single vertex. Within a precolored set (i.e., a set that is not uncolored), merge vertices of the same color. This merging operation may create parallel edges and self-loops in H . Discard all self-loops in H . Replace all parallel edges with a single weighted edge with weight equivalent to the number edges between the corresponding vertices. Edges between the vertices in G that are merged to the same vertex are treated as happy, as there is an optimal extended full coloring where the merged vertices are colored the same by Lemma 38. Clearly, H has at most $d + kd$ vertices in which t vertices are uncolored.

Now, MHE on G is converted to an instance of WEIGHTED MHE on H . By using Theorem 34, we can solve the instance of MHE on G in time $O^*(2^d)$. \square

Using similar arguments, we get the following results for MHV as well.

<p>MHV</p> <p>Input: A graph G, an integer k, a vertex subset $S \subseteq V(G)$, and a (partial) coloring $c : S \subseteq V(G) \rightarrow [k]$.</p> <p>Output: A coloring $\tilde{c} : V(G) \rightarrow [k]$ such that $\tilde{c} _S = c$ maximizing the number of happy vertices.</p>	<p>Parameter: neighborhood diversity t</p>
---	--

Lemma 40. *There is an optimal extended full coloring for an instance I of MHV such that each uncolored set P_i for $1 \leq i \leq d$ is monochromatic.*

Proof. Consider any optimal extended full coloring for an instance I . Suppose the vertices in a set P_i belong to more than one color class. Let Q_1 and Q_2 be the (disjoint and non-empty) sets of vertices of P_i belonging to color classes C_1 and C_2 , respectively. Let X_1 and X_2 be the neighbors of the vertices in Q_1 and Q_2 in color classes C_1 and C_2 , respectively, as shown in Figure 5.2. Without loss of generality, let us assume that $|X_1| \leq |X_2|$. By recoloring vertices in Q_1 with the color of C_2 , we retain an optimal solution without disturbing the colors of other vertices. If $|E(Q_1, Q_2)|$ is the number of edges between Q_1 and Q_2 , the gain in the number of happy edges by recoloring Q_1 is $|E(Q_1, Q_2)| + |Q_1|(|X_2| - |X_1|)$, which is strictly positive if P_i is a clique and non-negative if P_i is an independent set.

In conclusion, we have shown that every optimal extended full coloring makes each P_i inducing a clique monochromatic. Moreover, there is an optimal extended full coloring making each P_i inducing an independent set monochromatic. \square

Theorem 41. *For any $k \geq 1$, MHV can be solved in time $O^*(2^d)$, where d is the neighborhood diversity of the input graph.*

Proof. First we construct a weighted graph H from G as follows: merge each uncolored set into a single vertex. Within a precolored set (i.e., a set that is not uncolored), merge

vertices of the same color. This merging operation may create parallel edges and self-loops in H . Discard all self-loops in H . Replace all parallel edges with a single weighted edge with weight equivalent to the number edges between the corresponding vertices. Edges between the vertices in G that are merged to the same vertex are treated as happy, as there is an optimal extended full coloring where the merged vertices are colored the same by Lemma 38. Clearly, H has at most $d + kd$ vertices in which d vertices are uncolored.

Now, MHV on G is converted to an instance of WEIGHTED MHV on H . By using Theorem 34, we can solve the instance of MHV on G in time $O^*(2^d)$. \square

Chapter 6

Algorithm for Replacement Paths Problem

A *Replacement Shortest Path* (RSP) for the edge e_i (respectively, node v_i) is a shortest $s - t$ path in $G \setminus e_i$ (respectively, $G \setminus v_i$). The EDGE REPLACEMENT PATH problem is to compute RSP for all $e_i \in P_G(s, t)$. Similarly, the NODE REPLACEMENT PATH problem is to compute RSP for all $v_i \in P_G(s, t)$.

Like in all existing algorithms for RSP problem, our algorithm has two phases:

1. Computing shortest path trees rooted at s and t , T_s and T_t respectively.
2. Computing RSP using T_s and T_t .

For graphs with non-negative edge weights, computing an SPT takes $O(m + n \log n)$ time, using the standard Dijkstra's algorithm [28] using Fibonacci heaps [36]. However, for integer weighted graphs (RAM model) [75], planar graphs [44] and minor-closed graphs [74], $O(m + n)$ time algorithms are known. In this paper, to compute SPTs T_s and T_t (phase (i)) we use the existing algorithms. For phase (ii), we present an $O(m + l^2)$ time algorithm which is simple and easy to implement.

6.1 Edge Replacement Paths

We start by computing shortest path trees T_s and T_t . In the rest of the section we describe the algorithm for computing RSP using T_s and T_t (phase (ii)).

A potential replacement path for the edge $e_i = (v_{i-1}, v_i)$ can be seen as a concatenation of three paths A , B and C , where, $A = s \rightsquigarrow v_k \in P_G(s, t)$ for some $0 \leq k < i$, $B = v_k \rightsquigarrow v_r \in G \setminus E(P_G(s, t))$ for some $i \leq r \leq l$ and $C = v_r \rightsquigarrow t \in P_G(s, t)$ as shown in Figure 6.1. Here, the symbol \rightsquigarrow represents a path in G . One extreme case is when $|A| = 0$

and $|C| = 0$ (i. e. $v_k = s$ and $v_r = t$) as shown in Figure 6.1(b). Such a replacement path is also a potential replacement path for all the edges $e_i \in P_G(s, t)$. The other extreme case is when $|A| = i - 1$ and $|C| = l - i$ (i.e. $v_k = v_{i-1}$ and $v_r = v_i$) as shown in Figure 6.1(c). Such a replacement path is a potential replacement path only for the edge e_i .

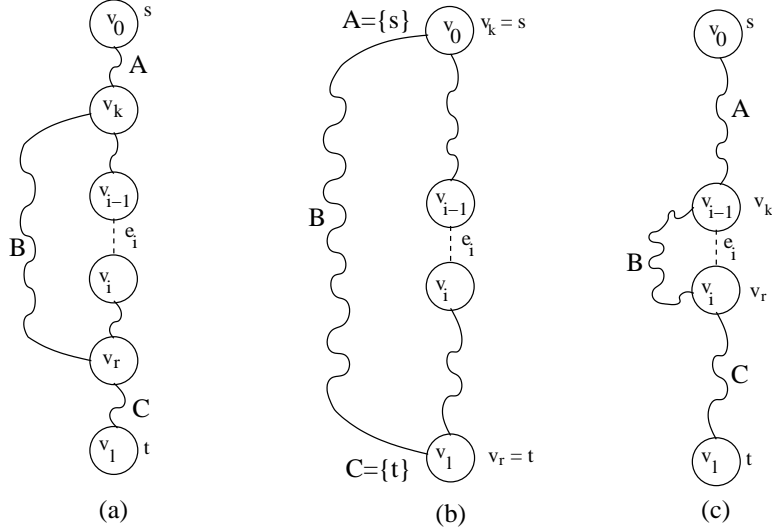


Figure 6.1: Potential replacement paths for the edge e_i . The zig-zag lines represent a path

Consider the shortest path tree rooted at s (T_s). When the edge $e_i = (v_{i-1}, v_i)$ is removed from T_s , T_s is disconnected into two sub-trees. $T_1(e_i)$ (sub-tree rooted at s) and $T_2(e_i)$ (sub-tree rooted at v_i). The vertex sets of $T_1(e_i)$ and $T_2(e_i)$ determine a cut in the graph G . Let $C(e_i)$ denote the set of all non-tree edges crossing the cut. These edges are called crossing edges, i.e., $C(e_i) = \{(x, y) \in E(G) \setminus e_i \mid x \in T_1(e_i) \wedge y \in T_2(e_i)\}$. In order to have a replacement path, the set $C(e_i)$ needs to be nonempty. And, any replacement path must use at least one crossing edge from $C(e_i)$. Moreover, as we see from the Lemmas 42 and 43 there exists an RSP that uses exactly one crossing edge.

Lemma 42 ([69]). *For all $(x, y) \in C(e_i)$, $d_{G-e_i}(s, x) = d_G(s, x)$ and $d_{G-e_i}(y, t) = d_G(y, t)$.*

Proof. If $(x, y) \in C(e_i)$, then $x \in T_1(e_i)$ and $y \in T_2(e_i)$. Shortest $s - x$ path is fully in $T_1(e_i)$ and does not include the edge e_i . Hence, $d_{G-e_i}(s, x) = d_G(s, x)$.

To prove $d_{G-e_i}(y, t) = d_G(y, t)$, for the sake of contradiction, let us assume that $d_{G-e_i}(y, t) \neq d_G(y, t)$ (i.e. $d_{G-e_i}(y, t) > d_G(y, t)$). It means that, $P_G(y, t)$ uses the edge $e_i = (v_{i-1}, v_i)$. This implies $d_G(v_i, y) > d_G(v_{i-1}, y)$. Since $y \in T_2(e_i)$, $d_G(v_{i-1}, y) > d_G(v_i, y)$ a contradiction. Hence, $d_{G-e_i}(y, t) = d_G(y, t)$. \square

Lemma 43 ([69]). *For any edge $e_i \in P_G(s, t)$, there exists a shortest $s - t$ path in $G - e_i$ which contains exactly one edge from $C(e_i)$.*

Proof. Let us consider a shortest $s - t$ path in $G - e_i$ (say P_1) which uses more than one crossing edge from $C(e_i)$. Let (x, y) be the last crossing edge in P_1 . Clearly $x \in T_1(e_i)$ and $y \in T_2(e_i)$. By replacing the part of P_1 from s to x , by the $s \rightsquigarrow x$ path in $T_1(e_i)$, we get a new path which is not longer than P_1 and uses exactly one edge from $C(e_i)$. \square

Using the Lemmas 42 and 43, we write the total weight of the RSP for the edge e_i as:

$$d_{G-e_i}(s, t) = \min_{(x', y') \in C(e_i)} \{d_G(s, x') + w(x', y') + d_G(y', t)\}. \quad (6.1)$$

All the terms in the equation (6.1) are available in constant time for a fixed (x', y') from T_s and T_t . Let (x, y) be the crossing edge that minimizes the RHS of the equation (6.1). We call that (x, y) the *swap edge*. If we have the swap edge, we can report the RSP as $s \rightsquigarrow x \rightarrow y \rightsquigarrow t$ in constant time. Every non-tree edge can be a potential crossing edge for every edge in $P_G(s, t)$. So, solving equation (6.1) by brute force gives us $O(ml)$ time algorithm. In this paper we present an $O(m + l^2)$ time algorithm. In the rest of the paper, we concentrate on computing the swap edge for each $e_i \in P_G(s, t)$.

6.1.1 Labeling the nodes of G

Every vertex of G is labeled with an integer value from 0 to l , with respect to the shortest path tree T_s . The process of labeling is as follows:

Let T_{v_i} be the sub-tree rooted at the node v_i in T_s . All the nodes in the sub-tree T_{v_l} are labeled with the integer value l . For $0 \leq i < l$, all the nodes in the sub-tree $T_{v_i} \setminus T_{v_{i+1}}$ are labeled with the integer value i . See Figure 6.2(a) for an example labeling.

Using pre-order traversal on T_s , we compute the labels of all the vertices in linear time. We start pre-order traversal from the source vertex s using zero as initial label. While visiting the children of a node recursively, the child node part of $P_G(s, t)$ (if any) will be visited last with an incremented label. Let $label(v)$ denote the label of a vertex v in G . The following Lemma is straightforward.

Lemma 44. *A non-tree edge $(x, y) \in C(e_i)$ if and only if $label(x) < i$ and $label(y) \geq i$. In other words, for a non-tree edge (x, y) , if $label(x) = i$ and $label(y) = i + r$ for some $r > 0$, then $(x, y) \in C(e_j)$, $\forall (i < j \leq i + r)$.*

6.1.2 Computing Swap Edges

We construct a directed acyclic graph which will aid us in computing the swap edges. We call this DAG as RSP-DAG, denoted by \widehat{G} . The following algorithm explains the construction of the RSP-DAG. An example RSP-DAG is shown in Figure 6.2(b).

Algorithm 7 Algorithm to construct the RSP-DAG $\widehat{G} = (\widehat{V}, \widehat{E})$.

```

1:  $\widehat{V} \leftarrow \emptyset$ 
2:  $\widehat{E} \leftarrow \emptyset$  ▷ Adding Nodes. Each node is identified by an ordered pair  $(i, j)$ 
3: for  $i = 0$  to  $l - 1$  do
4:   for  $j = i + 1$  to  $l$  do
5:      $\widehat{V} \leftarrow \widehat{V} \cup (i, j)$ 
6:   end for
7: end for ▷ Adding Edges
8: for each  $\widehat{u} = (i, j) \in \widehat{V}$  do
9:   if  $j - i > 1$  then
10:     $\widehat{E} \leftarrow \widehat{E} \cup ((i, j), (i, j - 1))$ 
11:     $\widehat{E} \leftarrow \widehat{E} \cup ((i, j), (i + 1, j))$ 
12:   end if
13: end for

```

Clearly, the number of vertices in the RSP-DAG is $O(l^2)$ and the number of edges is also $O(l^2)$. Every node has in-degree and out-degree at most two. The node with identifier $(0, l)$ has zero in-degree. Nodes $(i, i + 1), \forall (0 \leq i < l)$ have zero out-degree (sink nodes).

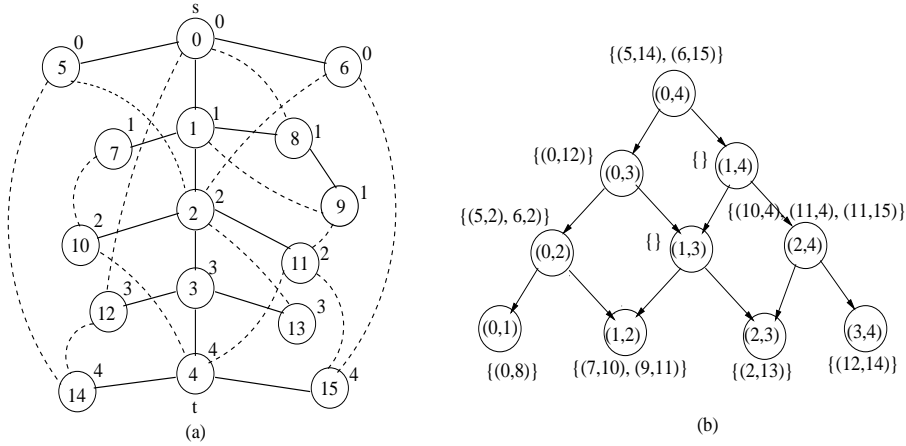


Figure 6.2: (a) An SPT rooted at s . Solid lines are part of the SPT. Dashed lines represent the non-tree edges (we omit the edge weights). Number inside the vertex circle denotes the vertex number, whereas the number above the vertex circle represents vertex label. (b) Corresponding RSP-DAG with set of non-tree edges associated with nodes

For each node $\widehat{u} = (i, j) \in \widehat{V}$, we associate a set $E_{(i,j)}$ of crossing edges. This set includes all the non-tree edges (x, y) such that $label(x) = i$ and $label(y) = j$. This association of crossing edges partitions the crossing edges into disjoint sets.

Lemma 45. *If the swap edge (x, y) for the tree edge $e_i \in P_G(s, t)$ is present in the edge set $(E_{(j,k)})$ of a node $\widehat{u} = (j, k) \in \widehat{V}$, then there exists a directed path from the node \widehat{u} to the node $\widehat{w} = (i - 1, i) \in \widehat{V}$ in the RSP-DAG.*

Proof. Clearly $j \leq i - 1$ and $k \geq i$, otherwise, (x, y) will not be the crossing edge for e_i . If \hat{u} is a sink node ($\hat{u} = \hat{w}$) in the RSP-DAG, then the theorem is trivially true.

Otherwise, if we observe the way edges are added in the RSP-DAG, for the node $\hat{u} = (j, k) \in \hat{V}$, two directed edges $((j, k), (j, k - 1))$ and $((j, k), (j + 1, k))$ are added and from these nodes, we keep adding edges to the lower level nodes in the RSP-DAG. We will eventually connect to the leaf node $\hat{w} = (i - 1, i) \in \hat{V}$. Hence there is a directed path from \hat{u} to \hat{w} . \square

Now we make a BFS traversal on the RSP-DAG starting from the node with identifier $(0, l)$. During the traversal, at every node, the minimum cost non-tree edge (x, y) (cost being $d(s, x) + w(x, y) + d(y, t)$) from the corresponding edge set is inserted into the edge sets of its two children. By the end of this process, minimum cost non-tree edges in the respective sink nodes give us the swap edges.

Theorem 46. *There is an algorithm for the EDGE REPLACEMENT PATH problem that runs in $O(T_{SPT}(G) + m + l^2)$ time using $O(m + l^2)$ space.*

Proof. $T_{SPT}(G)$ represents the time to compute SPTs T_s and T_t . Construction of the RSP-DAG takes $O(m + l^2)$ time and $O(m + l^2)$ space. BFS traversal on the RSP-DAG takes $O(l^2)$ time. During the traversal at each node $(i, j) \in \hat{V}$, we extract the minimum cost non-tree edge from the set of size at most $|E_{(i,j)}| + 2$. Time complexity of overall edge extraction steps is: $\sum_{i < j} |E_{(i,j)}| + 2 = O(m + l^2)$. Therefore the total time complexity is $O(T_{SPT}(G) + m + l^2)$. Space complexity is $O(m + l^2)$ which is the space to store the RSP-DAG. \square

Using the linear time algorithms for SPT, for integer weighted graphs, minor closed graphs our algorithm takes $O(m + l^2)$ time.

6.2 Node Replacement Paths

When the node $v_i \in P_G(s, t)$ is removed, the SPT T_s is partitioned as: $T_1(v_i)$ (sub-tree rooted at s), $T_2(v_i)$ (sub-tree rooted at v_{i+1}) and $F(v_i)$ (the remaining forest $T_s \setminus \{T_1(v_i) \cup T_2(v_i) \cup v_i\}$). The crossing edges are denoted as:

$$C'(v_i) = \{(x, y) \in E(G) \mid x \in T_1(v_i) \wedge y \in T_2(v_i)\} \quad (6.2)$$

$$C''(v_i) = \{(x, y) \in E(G) \setminus (v_i, v_{i+1}) \mid x \in F(v_i) \wedge y \in T_2(v_i)\} \quad (6.3)$$

$$C(v_i) = C'(v_i) \cup C''(v_i) \quad (6.4)$$

Lemma 47 ([69]). *For all $x \in T_1(v_i)$, $d_{G-v_i}(s, x) = d_G(s, x)$, and for all $y \in T_2(v_i)$, $d_{G-v_i}(y, t) = d_G(y, t)$.*

Proof. We omit the proof as the proof is similar to lemma 42 □

Using Lemma 47, the length of the RSP is written as:

$$d'_{G-v_i}(s, t) = \min_{(x,y) \in C'(v_i)} \{d_G(s, x) + w(x, y) + d_G(y, t)\} \quad (6.5)$$

$$d''_{G-v_i}(s, t) = \min_{(x,y) \in C''(v_i)} \{d_{G-v_i-T_2(v_i)}(s, x) + w(x, y) + d_G(y, t)\} \quad (6.6)$$

$$d_{G-v_i}(s, t) = \min\{d'_{G-v_i}(s, t), d''_{G-v_i}(s, t)\} \quad (6.7)$$

Having T_s and T_t , all the terms in the equations (6.5) and (6.6) are available in constant time, except the distance $d_{G-v_i-T_2(v_i)}(s, x)$ for $x \in F(v_i)$ (partial shortest path distance). We need all the partial shortest path distances $d_{G-v_i-T_2(v_i)}(s, x)$, $\forall v_i \in P_G(s, t)$ and $\forall x \in F(v_i)$.

To compute all the partial shortest path distances, we use the technique used in [62] and [60].

Let G_i (corresponding to the vertex v_i) be the graph constructed from G as follows: The vertex set of G_i , $V(G_i)$, consists of the source vertex s and the vertices which are part of the forest $F(v_i)$. The edge set of G_i , $E(G_i)$, consists of the following edges:

- Edges between the nodes within the forest $F(v_i)$. These edges will get the same edge weight as in G .
- For every $v \in F(v_i)$, an edge (s, v) is added whenever there is at least one edge from $T_1(v_i)$ to v . The weight of this edge is calculated as follows:

$$\tilde{w}(s, v) = \min_{(u,v) \in E(T_1(v_i), v)} \{d_G(s, u) + w(u, v)\} \quad (6.8)$$

That is,

$$V(G_i) = \{V(F(v_i))\} \cup \{s\} \quad (6.9)$$

$$E(G_i) = \{E(F(v_i), F(v_i))\} \cup \{(s, v) | (v \in F(v_i) \wedge E(T_1(v_i), v) \neq \emptyset)\} \quad (6.10)$$

G_i is a graph minor of G , since it can be obtained by edge contraction. Hence, SPT, $T_i(s)$ of G_i , rooted at s can be constructed in $T_{SPT}(G_i)$ time. Moreover, $d_{G-v_i-T_2(v_i)}(s, x) = d_{G_i}(s, x)$ for any $x \in F(v_i)$. As $F(v_i) \cap F(v_j) = \emptyset$, for any $i \neq j$, $V(G_i) \cap V(G_j) = \{s\}$. Construction of G_i and $T_i(s)$ for all i takes a total time of $O(\sum_{i=1}^{l-1} (T_{SPT}(G_i))) = O(T_{SPT}(G))$. $d_{G-v_i-T_2(v_i)}(s, x)$ for any $x \in F(v_i)$ is available in constant time from $T_i(s)$ of G_i .

Instead of computing $l - 1$ SPTs, $T_i(s)$, for all $1 \leq i \leq l - 1$, we compute one graph, $\tilde{G} = \bigcup_{i=1}^l G_i$, where G_i is constructed as explained earlier. \tilde{G} can be constructed from

G in $O(m + n)$ time. Single source shortest path tree rooted at s , \tilde{T}_s of \tilde{G} is computed in $O(T_{SPT}(\tilde{G})) = O(T_{SPT}(G))$ time. $d_{G-v_i-T_2(v_i)}(s, x)$ for any $x \in F(v_i)$ is available in constant time from \tilde{T}_s of \tilde{G} . Moreover, as $C''(v_i) \cap C''(v_j) = \emptyset, \forall(i \neq j)$, the distances $d''_{G-v_i}(s, t)$ for all v_i are available in linear time.

To compute $d'_{G-v_i}(s, t)$ for all v_i , we use the RSP-DAG. We use the vertex labeling on T_s (as computed in Section 6.1.1), for a non-tree edge (x, y) , $(x, y) \in C'(v_i)$ if and only if $label(x) < i$ and $label(y) > i$. In other words, for a non-tree edge (x, y) , if $label(x) = i$ and $label(y) = i + r$ for some $r > 1$, then $(x, y) \in C'(v_j)$, for all $i < j < i + r$.

Hence, the crossing edges $C'(v_i)$ will be part of edge sets associated with the vertices $(i, i + r), r > 1$ in the RSP-DAG. After the BFS traversal on the RSP-DAG, the minimum cost crossing edge (over $C'(v_i)$) for v_i is available in the edge set of the node $(i - 1, i + 1)$ in the RSP-DAG. We do not need to perform the BFS traversal on the RSP-DAG again, because, the data populated during the BFS traversal for the edge replacement paths suffices.

If we have the swap edge (x, y) for the vertex v_i , we can report the RSP in constant time as $s \rightsquigarrow x \rightarrow y \rightsquigarrow t$. Here $s \rightsquigarrow x$ is available from T_s if $(x, y) \in C'(v_i)$. It is constructed from SPTs T_s and \tilde{T}_s if $(x, y) \in C''(v_i)$.

Theorem 48. *There is an algorithm for the NODE REPLACEMENT PATH problem that runs in $O(T_{SPT}(G) + m + l^2)$ time using $O(m + l^2)$ space.*

Proof. $T_{SPT}(G)$ represents the time to compute SPTs T_s and T_t . Computing the distances $d''_{G-v_i}(s, t)$ for all v_i takes $O(T_{SPT}(G) + m + n)$ time. Computing $d'_{G-v_i}(s, t)$ for all v_i using the RSP-DAG takes $O(m + l^2)$ time and $O(m + l^2)$ space. Therefore the total time complexity is $O(T_{SPT}(G) + m + l^2)$. Space complexity is $O(m + l^2)$ which is the space necessary to store the RSP-DAG. \square

Using the linear time algorithms for SPT, for integer weighted graphs, minor closed graphs our algorithm takes $O(m + l^2)$ time.

From Theorems 46 and 48 we state the following Theorem.

Theorem 49. *There is an algorithm for the edge and the node replacement path problems that runs in $O(T_{SPT}(G) + m + l^2)$ time using $O(m + l^2)$ space.*

Chapter 7

Conclusions and Future Work

In this thesis we presented parameterized algorithms for the graph partitioning problems, MATCHING CUT, H -Free Coloring and Happy Coloring. We also studied the complexity of Happy Coloring problems for some special graph classes like trees, bipartite graphs, split graphs and complete graphs. We also presented a simple algorithm for replacement paths problem. The following are some of the open problems related to these problems.

For the MATCHING CUT problem, we presented an $O^*(2^{O(t)})$ time algorithm, where t is the tree-width of the graph. It is interesting to study the complexity of the MATCHING CUT problem parameterized by the size of the cut. That is given an undirected graph G and a positive integer ℓ the question is: does the graph G have a matching cut such that the number of edges in the cut is at most ℓ ?

When the graph G has degree at least 2, the MATCHING CUT problem in G is equivalent to the problem of deciding whether the line graph of G denoted by $L(G)$ has an independent vertex cut. The MATCHING CUT problem parameterized by the size of the cut is equivalent to the problem of deciding whether the line graph of G has independent vertex cut of size at most ℓ . The maximum independent set problem on line graphs is polynomial-time solvable, but we need an independent set $I \subseteq V(L(G))$ such that $|I| \leq \ell$ and I is a vertex cut in $L(G)$.

For the MATCHING CUT and H -FREE COLORING problems, we presented explicit combinatorial algorithms parameterized by the tree-width. The question is: are these algorithms optimal? Can we have ETH/SETH based lower bounds for the MATCHING CUT and H -FREE COLORING problems parameterized by tree-width?

For the MHV problem Agrawal [3] gave a kernel of size $O(k^2\ell^2)$. In this thesis, we obtained an $O(k + \ell)$ kernel for the MHE problem. It would be interesting to see if MHV problem admits an $O(k + \ell)$ kernel. For a arbitrary k , the MHE problem is NP-hard for planar graphs. It is interesting to study the complexity of the MHV problem for planar graphs.

In this thesis, we have shown that both MHE and MHV are in FPT with respect to the combined parameter $k + t$, where k is the number of colors used in the pre-coloring and t is the tree-width of the graph. The complexity of the MHE and MHV problems with respect to the parameter tree-width alone can be explored.

Bibliography

- [1] The multi-multiway cut problem. *Theoretical Computer Science*, 377(1):35–42, 2007.
- [2] Demetrios Achlioptas. The complexity of g -free colourability. *Discrete Mathematics*, 165-166(Supplement C):21–30, 1997.
- [3] Akanksha Agrawal. On the parameterized complexity of happy vertex coloring. Accepted to IWOCA 2017.
- [4] N. R. Aravind, Subrahmanyam Kalyanasundaram, and Anjeneya Swami Kare. Linear time algorithms for happy vertex coloring problems for trees. In *Proceedings of the 27th International Workshop on Combinatorial Algorithms (IWOCA)*, volume 9843 of *Lecture Notes in Computer Science*, pages 281–292. Springer, 2016.
- [5] N. R. Aravind, Subrahmanyam Kalyanasundaram, and Anjeneya Swami Kare. On structural parameterizations of the matching cut problem. In *Combinatorial Optimization and Applications*, pages 475–482, 2017.
- [6] N. R. Aravind, Subrahmanyam Kalyanasundaram, Anjeneya Swami Kare, and Juho Lauri. Algorithms and hardness results for happy coloring problems. *CoRR*, abs/1705.08282, 2017. URL <http://arxiv.org/abs/1705.08282>.
- [7] N. R. Aravind, Subrahmanyam Kalyanasundaram, and Anjeneya Swami Kare. Bipartitioning problems on graphs with bounded tree-width. *CoRR*, abs/1804.04016, 2018. URL <https://arxiv.org/abs/1804.04016>.
- [8] Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. Degree-constrained decompositions of graphs: Bounded treewidth and planarity. *Theoretical Computer Science*, 355(3):389 – 395, 2006.
- [9] Omer Berkman, Baruch Schieber, and Uzi Vishkin. Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values. *Journal of Algorithms*, 14(3):344–370, 1993.

- [10] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set Partitioning via Inclusion-Exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009.
- [11] Paul Bonsma. The complexity of the matching-cut problem for planar graphs and other graph classes. *Journal of Graph Theory*, 62(2):109–126, 2009.
- [12] Mieczysław Borowiecki and Katarzyna Jesse-Józefczyk. Matching cutsets in graphs of diameter 2. *Theoretical Computer Science*, 407(1-3):574–582, 2008.
- [13] Andreas Brandstädt, Feodor F. Dragan, Van Bang Le, and Thomas Szymczak. On stable cutsets in graphs. *Discrete Applied Mathematics*, 105(1):39–50, 2000.
- [14] Yixin Cao, Jianer Chen, and J.-H. Fan. An $O^*(1.84^k)$ parameterized algorithm for the multiterminal cut problem. *Information Processing Letters*, 114(4):167–173, 2014.
- [15] Yair Caro and Raphael Yuster. Graph decomposition of slim graphs. *Graphs and Combinatorics*, 15(1):5–19, 1999.
- [16] Guantao Chen and Xingxing Yu. A note on fragile graphs. *Discrete Mathematics*, 249(1-3):41–43, 2002.
- [17] Guantao Chen, Ralph J. Faudree, and Michael S. Jacobson. Fragile graphs with small independent cuts. *Journal of Graph Theory*, 41(4):327–341, 2002.
- [18] Sunil Chopra and M. R. Rao. On the multiway cut polyhedron. *Networks*, 21(1):51–89, 1991.
- [19] V. Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8(1):51–53, 1984.
- [20] D.G. Corneil and J. Fonlupt. Stable set bonding in perfect graphs and parity graphs. *Journal of Combinatorial Theory, Series B*, 59(1):1 – 14, 1993.
- [21] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.
- [22] Bruno Courcelle. The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. *Theoretical Informatics and Applications*, 26:257–286, 1992.
- [23] Bruno Courcelle. The monadic second order logic of graphs vi: on several representations of graphs by relational structures. *Discrete Applied Mathematics*, 54(2):117–149, 1994.

- [24] L. J. Cowen, R. H. Cowen, and D. R. Woodall. Defective colorings of graphs in surfaces: Partitions into subgraphs of bounded valency. *Journal of Graph Theory*, 10 (2):187–195, 1986.
- [25] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- [26] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiway cuts (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 241–251. ACM, 1992.
- [27] Xiaojie Deng, Bingkai Lin, and Chihao Zhang. Multi-multiway cut problem on graphs of bounded branch width. In *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, pages 315–324, 2013.
- [28] E W Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [29] David Easley and Jon Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, 2010.
- [30] Paul Erdős, Arthur L. Rubin , and Herbert Taylor. Choosability in graphs. In *In Proceedings of the West Coast Conference on Combinatorics, Graph Theory and Computing*, pages 125–157, 1979.
- [31] Arthur M. Farley and Andrzej Proskurowski. Networks immune to isolated line failures. *Networks*, 12(4):393–403, 1982.
- [32] Alastair Farrugia. Vertex-partitioning into fixed additive induced-hereditary properties is NP-hard. *The Electronic Journal of Combinatorics*, 11, 08 2004.
- [33] Jiří Fiala, Tomáš Gavenčiak, Dušan Knop, Martin Koutecký, and Jan Kratochvíl. Fixed Parameter Complexity of Distance Constrained Labeling and Uniform Channel Assignment Problems. In Thang N. Dinh and My T. Thai, editors, *Proceedings of the 22nd International Confence on Computing and Combinatorics (COCOON)*, volume 9797 of *Lecture Notes in Computer Science*, pages 67–78. Springer, 2016.
- [34] Fedor V Fomin and Dieter Kratsch. *Exact exponential algorithms*. Springer Science & Business Media, 2010.

- [35] Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.
- [36] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.
- [37] Robert Ganian. Using neighborhood diversity to solve hard problems. *CoRR*, abs/1201.3091, 2012. URL <http://arxiv.org/abs/1201.3091>.
- [38] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.
- [39] Luisa Gargano and Adele A. Rescigno. Complexity of conflict-free colorings of graphs. *Theoretical Computer Science*, 566:39–49, 2015.
- [40] Olivier Goldschmidt and Dorit S. Hochbaum. A polynomial algorithm for the k-cut problem for fixed k. *Math. Oper. Res.*, 19(1):24–37, 1994.
- [41] R. L. Graham. On primitive graphs and optimal vertex assignments. *Annals of the New York Academy of Sciences*, 175(1):170–186, 1970.
- [42] Branko Grünbaum. Acyclic colorings of planar graphs. *Israel Journal of Mathematics*, 14(4):390–408, 1973.
- [43] Magnús M. Halldórsson and Guy Kortsarz. Multicoloring: Problems and techniques. In *Mathematical Foundations of Computer Science 2004*, pages 25–41, 2004.
- [44] Monika R Henzinger, Philip Klein, Satish Rao, and Sairam. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997.
- [45] John Hershberger and Subhash Suri. Vickrey prices and shortest paths: What is an edge worth? In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, pages 252–259, 2001.
- [46] Saurabh Joshi, Subrahmanyam Kalyanasundaram, Anjeneya Swami Kare, and Sriram Bhyravarapu. On the tractability of (k, i) -coloring. In *Algorithms and Discrete Applied Mathematics*, pages 188–198. Springer International Publishing, 2018.
- [47] Anjeneya Swami Kare. A simple algorithm for replacement paths problem. *Electronic Notes in Discrete Mathematics*, 53:307–318, 2016. International Conference on Graph Theory and its Applications.

- [48] Anjeneya Swami Kare and Sanjeev Saxena. Efficient solutions for finding vitality with respect to shortest paths. In *6th IEEE International Conference on Contemporary Computing (IC3)*, pages 70–75, 2013.
- [49] Michał Karpiński. Vertex 2-coloring without monochromatic cycles of fixed size is NP-complete. *Theoretical Computer Science*, 659(Supplement C):88–94, 2017.
- [50] Sulamita Klein and Celina M. H. de Figueiredo. The NP-completeness of multi-partite cutset testing. *Congressus Numerantium*, 119:217–222, 1996.
- [51] Ton Kloks, editor. *Treewidth: Computations and Approximations*. Lecture Notes in Computer Science, Springer, 1994.
- [52] Dieter Kratsch and Van Bang Le. Algorithms solving the matching cut problem. *Theoretical Computer Science*, 609(2):328–335, 2016.
- [53] Stefan Kratsch and Pascal Schweitzer. Isomorphism for graphs of bounded feedback vertex set number. In *Algorithm Theory - SWAT 2010*, pages 81–92. Springer Berlin Heidelberg, 2010.
- [54] Ewa Kubicka, Grzegorz Kubicki, and Kathleen A. McKeon. Chromatic sums for colorings avoiding monochromatic subgraphs. *Electronic Notes in Discrete Mathematics*, 43:247–254, 2013.
- [55] Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012.
- [56] Michael Langberg, Yuval Rabani, and Chaitanya Swamy. Approximation algorithms for graph homomorphism problems. In *Proceedings of the 9th International Conference on Approximation Algorithms for Combinatorial Optimization Problems, and 10th International Conference on Randomization and Computation (APPROX-RANDOM)*, volume 4110 of *Lecture Notes in Computer Science*, pages 176–187. Springer, 2006.
- [57] Paul F Lazarsfeld and Robert K Merton. Friendship as a social process: A substantive and methodological analysis. *Freedom and Control in Modern Society*, 18(1):18–66, 1954.
- [58] Van Bang Le and Bert Randerath. On stable cutsets in line graphs. In *27th International Workshop Graph-Theoretic Concepts in Computer Science (WG 2001) Boltenhagen, Germany*, pages 263–271, 2001.
- [59] Van Bang Le, Raffaele Mosca, and Haiko Müller. On stable cutsets in claw-free graphs and planar graphs. *Journal of Discrete Algorithms*, 6(2):256–276, 2008.

- [60] Cheng-Wei Lee and Hsueh-I Lu. Replacement paths via row minima of concise matrices. *SIAM Journal on Discrete Mathematics*, 2(1):206–225, 2014.
- [61] Ewa M. Kubicka, Grzegorz Kubicki, and Kathleen A. McKeon. Chromatic sums for colorings avoiding monochromatic subgraphs. 43:541–555, 08 2015.
- [62] Jay Mahadeokar and Sanjeev Saxena. Faster replacement paths algorithms in case of edge or node failure for undirected, positive integer weighted graphs. *Journal of Discrete Algorithms*, 23:54–62, 2013.
- [63] K Malik, A K Mittal, and S K Gupta. The k most vital arcs in the shortest path problem. *Operations Research Letters*, 8:223–227, 1989.
- [64] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology*, 27(1):415–444, 2001.
- [65] Walter Meyer. Equitable coloring. *The American Mathematical Monthly*, 80(8): 920–922, 1973.
- [66] Neeldhara Misra and I. Vinod Reddy. The parameterized complexity of happy colorings. Accepted to IWOCA 2017.
- [67] Augustine M. Moshi. Matching cutsets in graphs. *Journal of Graph Theory*, 13(5): 527–536, 1989.
- [68] Enrico Nardelli, Guido Proietti, and Peter Widmayer. A faster computation of the most vital edge of a shortest path. *Information Processing Letters*, 79(2):81–85, 2001.
- [69] Enrico Nardelli, Guido Proietti, and Peter Widmayer. Finding the most vital node of a shortest path. *Theoretical Computer Science*, 296(1):167–177, 2003.
- [70] Maurizio Patrignani and Maurizio Pizzonia. The complexity of the matching-cut problem. In *27th International Workshop Graph-Theoretic Concepts in Computer Science (WG 2001) Boltenhagen, Germany*, pages 284–295, 2001.
- [71] Michaël Rao. MSOL partitioning problems on graphs of bounded treewidth and clique-width. *Theoretical Computer Science*, 377(1):260–267, 2007.
- [72] Neil Robertson and P.D Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.
- [73] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *J. ACM*, 44(4): 585–591, 1997.

- [74] Siamak Tazari and Matthias Muller-Hannemann. Shortest paths in linear time on minor-closed graph classes, with an application to steiner tree approximation. *Discrete Applied Mathematics*, 157(4):673–684, 2009.
- [75] Mikkel Thorup. Floats, integers, and single source shortest paths. *Journal of Algorithms*, 35(2):189–201, 2000.
- [76] Yongqi Wu, Jinjiang Yuan, and Yongcheng Zhao. Partition a graph into two induced forests. *Journal of Mathematical Study*, 1:1–6, 01 1996.
- [77] Mingyu Xiao and Hiroshi Nagamochi. Complexity and kernels for bipartition into degree-bounded induced graphs. *Theoretical Computer Science*, 659:72–82, 2017.
- [78] Peng Zhang and Angsheng Li. Algorithmic aspects of homophyly of networks. *Theoretical Computer Science*, 593:117–131, 2015.
- [79] Peng Zhang, Tao Jiang, and Angsheng Li. Improved Approximation Algorithms for the Maximum Happy Vertices and Edges Problems. In *Proceedings of the 21st Annual International Conference on Computing and Combinatorics (COCOON)*, volume 9198 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2015.